НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

«До захисту допущено»
В.О. завідувача кафедри
_____ О.Л. Тимощук

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 “Комп’ютерні науки”

на тему: «Система розпізнавання команд за допомогою читання з губ»

Виконав:
студент IV курсу, групи КА-55
Дровальов Антон Андрійович _____

Керівник:
доц. кафедри ММСА,
доцент, к.т.н. Дідковська М.В. _____

Консультант з економічного розділу:
доцент, к.е.н. Шевчук О. А. _____

Консультант з нормоконтролю:
доцент, к.т.н. Коваленко А.Є. _____

Рецензент:
доц. кафедри ПЗКС
факультету прикладної математики
НТУУ «КПІ ім. І. Сікорського»,
доцент, к.т.н. Заболотня Т.М. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки – 6.050101 "Комп'ютерні науки"

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.Л. Тимощук
(підпис)
«___»_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту
Дровальову Антону Андрійовичу

1. Тема роботи Система розпізнавання команд за допомогою читання з губ, керівник роботи Дідковська Марина Віталіївна, доцент кафедри ММСА.

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи _____
_____
_____
_____

4. Зміст роботи _____
_____
_____
_____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) _____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка

Студент _____ А.А. Дровальов
(підпис)

Керівник роботи _____ М.В. Дідковська
(підпис)

РЕФЕРАТ

Дипломна робота: 110 с., 28 рис., 7 табл., 2 додатки, 25 джерел.

НЕЙРОННА МЕРЕЖА, ЧИТАННЯ З ГУБ, РОЗПІЗНАВАННЯ КОМАНД, РОЗПІЗНАВАННЯ ОБЛИЧЧЯ

Метою роботи є розробка додатку, що зчитує команди, сказані людиною, з камери в реальному часі, та вивід цих команд на екран. У дослідженні проаналізовано методи глибокого навчання, та застосовано деякі математичні підходи.

Результати роботи:

- розроблено додаток, що працює в режимі реального часу
- розроблено алгоритм детекції пауз між різними фразами
- реалізовано інтерфейс користувача для даного додатку

Додаток використовується у сферах, де немає можливості застосувати аудіо-обробку.

ABSTRACT

Theme of work: "System of commands recognition using lipreading".

NEURAL NETWORK, LIPREADING, RECOGNIZING COMMANDS, RECOGIZING OF FACE

Thesis contains 110 p., 28 fig., 7 tabl., 2 appendices and 25 sources.

The purpose of the work is to develop an application that reads the commands spoken by a person from the camera in real time, and displays these commands on the screen, or perform actions that correspond to them. The study analyzed the methods of deep learning, and applied some mathematical approaches.

Results of work:

- A real-time application is developed
- an algorithm for detecting pauses between different phrases has been developed
- The user interface for this application is implemented

The application is used in areas where it is not possible to apply audio processing.

ЗМІСТ

ВСТУП

У наш час дуже актуальна задача розпізнавання компю’тером людських команд. Існує багато алгоритмів розпізнавання голосу, жестів, рухів. Але також існує багато проблем, пов“язаних з неможливістю зчитування звуку комп'ютером, або з вадами людини. Тому, постає актуальна для нашого часу задача, що полягає в розпізнаванні команд без звуку, тобто розпізнавання мови з губ. Наразі не існує готового додатку, що розпізнавав би мову лише по відеозапису. Тому автоматизація зчитування слів та речень з губ значно допоможе великій частині людей. Раніше було запропоновано багато досліджень в цьому напрямку, але усі підходи не передбачують повне розпізнавання з губ послідовності речень, а лише окремо букв, або слів. Також існують підходи з частковим розпізнаванням, тобто використовується як візуальна складова, так і аудіо.

Метою дослідження є розробка додатку, що буде зчитувати команди, сказані людиною, з камери в реальному часі.

У запропонованому підході вхідними даними є відеозапис, або відеопотік в реальному часі, після чого система аналізує потік на наявність фрази, якщо фраза виявлена, то вона подається на обробку і ми отримуємо вихідні дані у вигляді текстової команди, чи дії, що відповідає цій команді.

Даний додаток є корисним для багатоьх областей. Він може використовуватись у дуже зашумленних приміщеннях, чи місцях, де неможливо подати команду за допомогою звуку. Наприклад, на заводі, де у людині зайняті руки, що унеможливлює подати команду жестами, а навколо дуже шумно. Також додаток є корисним для німих людей, та для людей з вадами слуху.

Розпізнавання команд за допомогою читання з губ — завдання розшифровки тексту з руху роту спікера. Традиційні підходи розділили цю проблему на два етапи: вивчення візуальних особливостей, та прогнозування. Тим не менш, існуючі роботи виконують лише класифікацію слів, а не передбачення послідовності на рівні команди чи речення.

Системи розпізнавання мови можна умовно розділити на кілька класів по тому, які типи послідовностей слів вони здатні аналізувати:

а) Окремі слова;

Такі системи спираються на те, що вимова кожного слова буде оточена тишею з обох сторін, тобто змушують мовця робити паузи між сусідніми входженнями. Виділення пауз в мові видається окремим завданням. У такому формулюванні завдання розпізнавання можна розглядати як задача класифікації

б) Зв'язна мова;

Деяким входженням слів дозволяється йти один за одним з мінімальною паузою

в) Злитна мова;

Практично природна мова людини. Одна з найскладніших завдань розпізнавання

У даній роботі розглядається клас, що відповідає другому пункту - розпізнавання слів невеликого словника на записах, що містять кілька вимовлених слів поспіль.

РОЗДІЛ 1 АНАЛІЗ ЗАДАЧІ РОЗПІЗНАВАННЯ МОВИ ЗА ВІЗУАЛЬНИМИ ХАРАКТЕРИСТИКАМИ

1.1 Аналіз актуальності задачі

Читання з губ відіграє вирішальну роль у людському спілкуванні і розумінні мови. Це важке завдання для людей, особливо за відсутності контексту. Наприклад, Фішер дає 5 категорій візуальних фонем , які зазвичай плутаються людьми при перегляді рота спікера.

Віземи - аналогічні рухи губ, що складаються з алфавіту для читання з губ. А саме, декілька звуків мають однакову форму. Існує ступінь неоднозначності між приголосними, ця проблема, добре задокументована Фішером у його обширному дослідженні візем, показана на рисунку 1.1 :

Рисунок 1.1 - Віземи

Це зображення показує, що деякі звуки, такі як «p» і «b», мають однаковий вигляд, що означає, що відповідні фонеми дуже важко визначити для людини. Наприклад, вислови "elephant juice" на губах однакові з "I love you".

Отже, у людини дуже мала здатність читання з губ. Люди з вадами слуху досягають точності лише 17 ± 12% навіть для обмеженої підмножини 30 односкладних слів, та 21 ± 11% для 30 складних слів .Тому важливою метою є автоматизація розпізнавання тексту з рухів губ. Автоматизоване читання з губ володіє величезним практичним потенціалом, це можна використовувати для поліпшення слухових апаратів, діагностикою в громадських місцях, безпекою, розпізнаванням мови в шумному оточенні, біометричною ідентифікацією та обробкою фільмів.

Системи засновані на візуальних ознаках можуть використовуватися для аутентифікації, реалізації інтерфейсів введення інформації або управління. Останнє особливо актуально в зв'язку з широким розповсюдженням мобільних пристроїв, використання яких часто відбувається в зашумлених умовах, сильно знижують якість розпізнавання аудіо-сигналу. Також даний підхід може використовуватися у випадках, коли людина з якихось причин не має можливості говорити вголос.

Однак розпізнавання мови, засноване на візуальної інформації в загальному випадку складніше аналізу аудіо-сигналу. Людська мова містить близько 50 фонем (мінімальна помітна одиниця аудіо-потоку) в той час як по губах можливо розрізнити близько 10-15 візем (груп візуально нерозрізних фонем). Таким чином, послідовність візем часто може не відповідати конкретному слову і точність читання по губах сильно залежить від контексту. Крім того, навіть серед людей які розмовляють на одному діалекті відповідність між рухами губ і сказаними віземами може дуже сильно відрізнятися, що робить майже неможливим побудову загальної відео-моделі розпізнавання без апріорної інформації про "стилі" руху губ людини.

Машинне розпізнавання з губ доволі важке, оскільки це вимагає виявлення просторових особливостей від відео (так як важливі як положення, так і рух). Однак більшість існуючих робіт виконує лише класифікацію слів, але не їх послідовності.

Модель машинного навчання, що використовувалася в даній роботі, працює на рівні символів, використовуючи згорткові нейронні мережі, рекурентні нейронні мережі, а також CTC.

Дана модель дає результати по корпусу GRID - одному з небагатьох загальнодоступних вибірок даних, що досягають точності розпізнавання на рівні речення 95%. Наприклад, одна з кращих точностей була досягнута при класифікації слів цього набору даних - 86,4%.

1.2 Аналіз існуючих підходів

Більшість підходів до розпізнавання мови можна розбити на наступні послідовні кроки:

а) Препроцесінг - включає виділення відрізків мовлення;

б) Виявлення ознак;

в) Декодування;

Власне, розшифровка сказаної інформації. Відбувається з використанням:

1) Акустичної моделі

Описує залежність між аудіо-сигналом і одиницями мови (майже завжди фонемами)

2) Словника

Безліч вимовлених слів разом з їх транскрипціями

3) Мовні моделі

Розподіл ймовірностей над множинами всіх речень або окремих слів

г) Постпроцесінг

На виході попереднього пункту може вийти набір можливих послідовностей слів разом з їх оцінками (наприклад, вірогідностями). При

виборі остаточної відповіді можна використовувати будь-які високорівневі вимоги, які не було можливості прийняти до уваги на описаних раніше етапах.

Схематично процес можна зобразити , як показано на рисунку 1.2 :

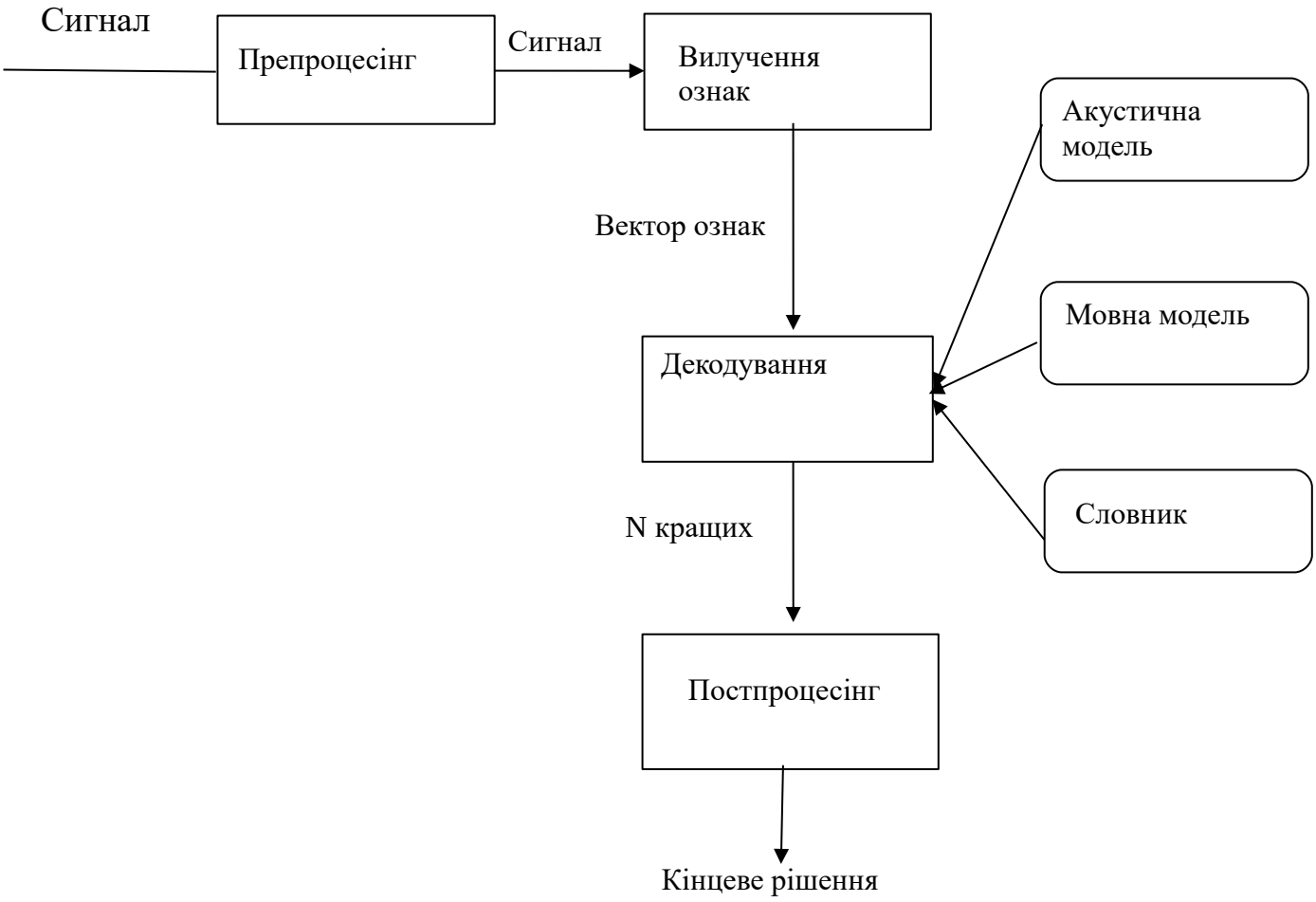


Рисунок 1.2 – Загальна схема обробки

Набори даних:

Переважна кількість вільно доступних наборів даних з розміченими аудіо та відеозаписами є на англійській мові. Серед них можна виділити декілька найпопулярніших: (AVICar, AVLetters, AVLetters2, BBC TV, CUAVE, OuluVS1, OuluVS2) , але більшість містить лише окремі слова або занадто малі. Єдиним виключенням є корпус GRID який має аудіо- та відеозаписи спікерів, по 1000 речень кожен, загальний час записів 28 годин, загалом містить 34000 речень.

Основні набори і їх характеристики:

а) GRID Містить 33 спікера, для кожного записано 1000 3-х секундних відео з вимовою 6 слів з різних класів.
б) CUAVE 36 спікерів, словник з 10 цифр
в) LILiR TwoTalk corpus 4 діалога по 12 хвилин між двома людьми
г) AVLetters1 10 спікерів, по 3 повторення кожної букви англійського алфавіту (всього 780 вимови)

В роботі використовувався набір даних GRID для оцінки моделі, оскільки він включає речення і має найбільшу кількість даних. Речення складаються з наступної простої граматики:
command(4)+color(4)+preposition(4)+letter(25)+digit(10)+adverb(4),
де число позначає, скільки слів є для кожної з 6 категорій. Категорії складаються, відповідно, з ,{bin, lay,place, set},{blue, green, red, white},{at, by, in, with},{A,...,Z}\{W},{zero, . . . , nine}, та {again, now, please, soon}, що дає можливість отримати 6000 можливих речень. Наприклад, можливими реченнями є “set blue by A four please” та “place red at C zero again”

1.3 Аналіз існуючих рішень

Окреслимо існуючі підходи до автоматизованого читання з губ. Існує багато підходів, які не застосовують машинне навчання. Такі підходи вимагають важкої попередньої обробки кадрів для вилучення ознак з відео(наприклад , оптичного потоку, чи виявлення руху)

Так як виникали доволі великі обмеження на задачу розпізнавання мови з відеопотоку, ранні дослідження зосереджувалися на спрощеній версії проблеми. Спочатку, інженерні рішення використовували моделі розпізнавання обличчя, які розміщували обмежувальні лінії навколо рота, і визначали модель губ незалежно від орієнтації обличчя. Деякими поширеними ознаками були співвідношення ширина-висота обмежувальної рамки для

виявлення роту, зовнішній вигляд (інтенсивність пікселів на червоному каналі зображення) і наближення кількості зубів від «білизни» на зображенні .

Ці підходи отримали вражаючі результати (понад 70% точність слів) для тестів, які проводилися з класифікаторами, підготовленими на тому ж самому спікері, на якому вони були протестовані. Але продуктивність була сильно пошкоджена, коли намагалися читати з губ осіб, які не були включені в навчальний набір. Виявлення губ у чоловіків з вусами також було більш складним і, отже, показники на таких випадках були низькими. Тому, в кінцевому підсумку задачу не вдалося узагальнити добре.

Зокрема, Goldschen (1997) був першим, хто зробив візуальне розпізнавання тексту для речення, використовуючи приховані марковські моделі (HMM).

Пізніше, Neti (2000) першими зробили аудіовізуальне розпізнавання мови з речення за допомогою HMM, на наборі даних IBM ViaVoic. Автори доповнюють продуктивність розпізнавання мови в шумному оточенні шляхом злиття візуальних функцій з аудіо. Набір даних містить 17111 висловлень 261 оратора для навчання (близько 34,9 годин) і не є загальнодоступним. Як було сказано, їхні візуальні результати не можуть бути інтерпретовані як тільки візуальне розпізнавання, оскільки вони доповнюються аудіозвуками.

Крім того, Gergen (2016) досягли на наборі даних GRID 86,4%. Генералізація та вилучення особливостей руху для різних людей вважається відкритою проблемою, про що йдеться у (Zhou , 2014).

Класифікація з глибоким навчанням: В останні роки було зроблено декілька спроб застосувати глибоке навчання до читання з губ. Проте всі ці підходи виконують лише класифікацію слова або фонеми, тоді як в даній моделі виконуєтья повне передбачення послідовності речень. Підходи включають вивчення мультимодальних аудіовізуальних, вивчення візуальних особливостей як частини традиційного мовлення (наприклад, HMM, GMM-HMM і т.д.) для класифікації слів та / або фонем або їх комбінації. Багато з цих

підходів віддзеркалюють ранній прогрес у застосуванні нейронних мереж для акустичної обробки в розпізнаванні мови (Hinton, 2012).

Chung & Zisserman (2016a) пропонують просторові та просторово-часові згорткові нейронні мережі на основі VGG для класифікації слів. Архітектури оцінюються на наборі даних BBC TV (333 і 500 класів). Але, їхні моделі не можуть обробляти змінну довжину слів, і вони не намагаються передбачати послідовність на рівні речення.

Chung & Zisserman (2016b) навчають моделі за допомогою збігу аудіовізуального максимуму , які вони використовують як вхідні дані для LSTM для класифікації 10-фраз на наборі даних OuluVS2, але це також не можна вважати роботою, що базується виключно на візуальних даних.

Wand (2016) запроваджують рекурентні нейронні мережі LSTM для читання з губ.

Garg (2016) застосовують VGG - попередньо навчені на обличчях для класифікації слів і фраз з набору даних MIRACL-VC1, який має лише 10 слів і 10 фраз. Тим не менш, їх краща рекурентна модель навчається шляхом заморожування параметрів VGGNet, а потім навчання RNN, а не навчання спільно. Їхня найкраща модель досягає лише 56,0% точності класифікації слів, і 44,5% точності класифікації фрази, незважаючи на те, що обидві з них - завдання класифікації 10 класів.

Прогнозування послідовності слів в розпізнаванні мови: Область автоматичного розпізнавання мови (ASR) не була б у сьогоднішньому стані без сучасних досягнень глибокого навчання. Втрата тимчасової класифікації (CTC) від Graves (2006) зробила великий вклад в дану галузь. Як згадувалося раніше, багато недавніх робіт здіснили прогрес в ASR, але зупинялись на прогнозуванні послідовності .

Модель, що використовується в даній роботі, є першою, яка виконує прогнозування на рівні послідовності слів, для розпізнавання мови. Тобто модель приймає як вхідні дані послідовність зображень, а на виході отримуємо послідовність слів, або лексем.

Наведемо порівняльну характеристику моделей:

Таблиця 1.1 – Порівняльна характеристика існуючих методів

Method	Dataset	Size	Output	Accuracy
Fu et al. (2008)	AVICAR	851	Digits	37.9%
Hu et al. (2016)	AVLetter	78	Alphabet	64.6%
Papandreou et al. (2009)	CUAVE	1800	Digits	83.0%
Chung & Zisserman (2016a)	OuluVS1	200	Phrases	91.4%
Chung & Zisserman (2016b)	OuluVS2	520	Phrases	94.1%
Chung & Zisserman (2016a)	BBC TV	>400000	Words	65.4%
Gergen et al. (2016)	GRID	29700	Words*	86.4%
LipNet	GRID	28775	**Sentences**	**95.2%**

Як бачимо, більшість існуючих методів застосовується лише для невеликого набору даних. Багато рішень застосовується лише для цифр, або букв. Також Доволі багато рішень існує для окремих слів, або фраз. З даної таблиці можна зробити висновок, що найкраща модель є остання, яку й було застосовано для програмного продукту.

1.4 Формалізація постановки задачі дослідження

Розглянемо задачу, що необхідно вирішити. В даній дипломній роботі постає задача розробки програмного продукту, що має такі властивості:

а) Вхідними даними є відеозапис, або відеопотік в реальному часі.

б) Як результат програмний продукт повинен зробити передбачення, що було сказано на вхідному відеозаписі, чи відеопотоці.

Для того, щоб реалізувати даний програмний продукт, постають наступні задачі для роботи:

а) Аналіз існуючих моделей розпізнавання

б) Вибір моделі, що найкраще підходить для даних потреб

в) Вибір набору даних, на якому буде проведено дослідження

г) Дослідження характеристик відеозаписів даного набору даних

д) Дослідження характеристик відео в режимі реального часу

є) Реалізація функціоналу для програмного продукту

Задачі , які повинен вирішувати програмний продукт:

а) розпізнавання обличчя

б) розпізнавання губ на обличчі

в) обрахунок площі на зображенні, що займають губи

г) обчислення характеристик стану губ (в спокої, або ні)

д) алгоритм детектування пауз

є) застосування моделі для розпізнавання речень з отриманих кадрів

1.5 Висновки за розділом

В першому розділі було розглянуто актуальність задачі, проаналізовано існуючі підходи, та сформульовано вимоги до роботи, та програмного продукту.

Можна зробити висновок, що в наш час, представлена робота є актуальною, так як на даний момент часу проводиться багато досліджень з питань розпізнавання мови, використовуючи лише відеопотік. Дана галузь є достатньо новою, тому в світі не існує повністю готових рішень для розпізнавання мови, використовуючи читання з губ. Тому головною метою дипломної роботи була розробка програмного продукту для розпізнавання мови, а саме фраз, та речень, використовуючи лише візуальні дані. Так як існуючі рішення здатні розпізнавати лише окремі слова, або, в деяких випадках, фрази, було вирішено розробити алгоритм розділення мови на фрази, тобто розмежування. Це дає можливість використовувати навіть існуючі алгоритми для детектування простих фраз.

РОЗДІЛ 2 МЕТОДИ РОЗПІЗНАВАННЯ МОВИ ВИКОРИСТОВУЮЧИ ЧИТАННЯ З ГУБ

2.1 Аналіз математичних основ

У багатьох статтях про розпізнавання мови довгий час базовим підходом вважалися приховані марковські моделі , проте останнім часом активно розвиваються методи, засновані на застосуванні нейронних мереж, особливо рекурентних .

Більш детальний огляд зазначених методів:

а) Приховані марковські моделі

Прихована марковська модель (ПММ) представляє з себе систему, що імітує випадковий процес, еволюція якого залежить тільки від поточного стану моделі і не залежить від попередньої історії. В такому випадку постає завдання знаходження значень прихованих (невідомих) параметрів моделі при заданій послідовності спостережуваних значень. Таким чином марковский процес можна зобразити, як показано на рисунку 2.1:

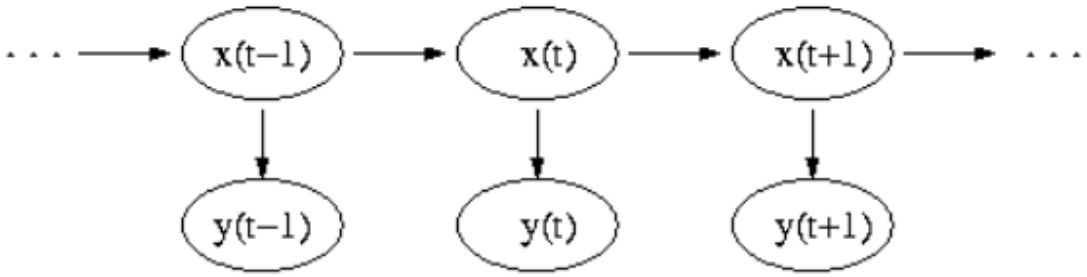


Рисунок 2.1 – Марковський процес

Де x_t є {1,. . . , N} - прихований стан моделі в момент часу t, $y_t \epsilon R^d$- змінна в момент часу t. Значення y_t залежить тільки від x_t, а x_t тільки від стану в момент часу t-1, тобто від x_{t-1} . Для подальшого завдання моделі потрібно

визначити матрицю вірогідності переходів a є $R^{N\,x\,N}$ | $a_{i,j}$ = p (x_t = j | x_{t-1} = i), розподілу спостерігаємих змінних для кожного стану p (y_t | x_t) і розподіл вірогідностей початкових станів (x_1). Зазвичай розподіл (y_t |x_t) =b_t роблять нормальним b_t (y) =N(y; μ_t ,Σt).

У загальному випадку ймовірнісні моделі застосовуються при розпізнаванні з губ наступним чином . Після вилучення векторів ознак з сигналу Y =y_1...y_t декодер намагається знайти послідовність слів w = $w_1 \ldots w_L$, якими з найбільшою ймовірністю сгенерована Y, іншими словами знаходить $w_{best} = argmax_w$ {P (w | Y)}. Часто незручно працювати з вірогідністю P (w | Y), тому, використовуючи формулу Байеса, її замінюють на:

$$argmax_w \left\{\frac{P(Y|w)\cdot P(w)}{P(Y)}\right\} = argmax_w\{P(Y|w)\cdot P(w)\} \qquad (2.1)$$

Тут P (Y | w) визначається акустичною моделлю, а P (w) — мовною. Кожному слову відповідає впорядкована послідовність фонем (можливо не одна, оскільки у слова може бути кілька правильних вимов). Акустична модель для конкретного слова виходить конкатинацією моделей для окремих фонем, які, в свою чергу, навчаються на розміченій тренувальної вибірці, наприклад, за допомогою алгоритму прямого-зворотного ходу . Можна уточнити формулу розрахунку P (Y | w) як сума по всіх можливих вимовам послідовності w :

$$P(Y|w) = \sum_Q P(Y|Q)\cdot P(Q|w) \qquad (2.2)$$

Перший множник P(Y|Q) при заданій марковській моделі для Q=$q_{(w1)}$, . . . , $q_{(wL)}$, отриманий конкатенацією моделей для своїх складових, вважається як сума вірогідностей по всім послідовностям станів Θ =θ_0, . . . , θ_{T+1}, а саме :

$$P(Y|Q) = \sum_\theta P(\theta, Y|Q) = \sum_\theta a_{\theta_0,\theta_1} \Pi_{t=1}^{T} b_{\theta_t}(y_t) a_{\theta_t,\theta_{t+1}} \qquad (2.3)$$

Стани θ0 і θT + 1 введені для зручності послідовної конкатенації моделей одна з одною і великої ролі не грають. Другий множник вважається як просто

добуток ймовірностей по кожному слову, що було вимовлено відповідним чином:

$$P(Q|w) = \Pi_{l=1}^{L} P(q^{(wl)}|w_l) \quad (2.4)$$

Однак, зазвичай описані моделі використовуються не для знаходження послідовності сказаних слів, а для розпізнавання фонем, тобто для отримання розподілу ймовірностей вимовлених фонем в кожен момент часу. Далі аналіз проводиться за допомогою язикової моделі алгоритмів, спеціалізованих для конкретного завдання.

б) Нейромережі

Нейронна мережа - це модель, яка використовується для оцінки невідомої функції f (x): Rd1 → Rd2, що залежить від великого числа параметрів. Вона являє собою набір взаємодіючих між собою так званих нейронів. Зв’язки між ними мають чисельні ваги, які можуть бути перераховані (навчені) для кращої відповідності заданим вхідним даними. Така модель вперше з'явилася в 40-х роках минулого століття, натхненна базовими принципами роботи людського мозку.

У загальному випадку нейронна мережа може бути візуалізована, як показано на рисунку 2.2:

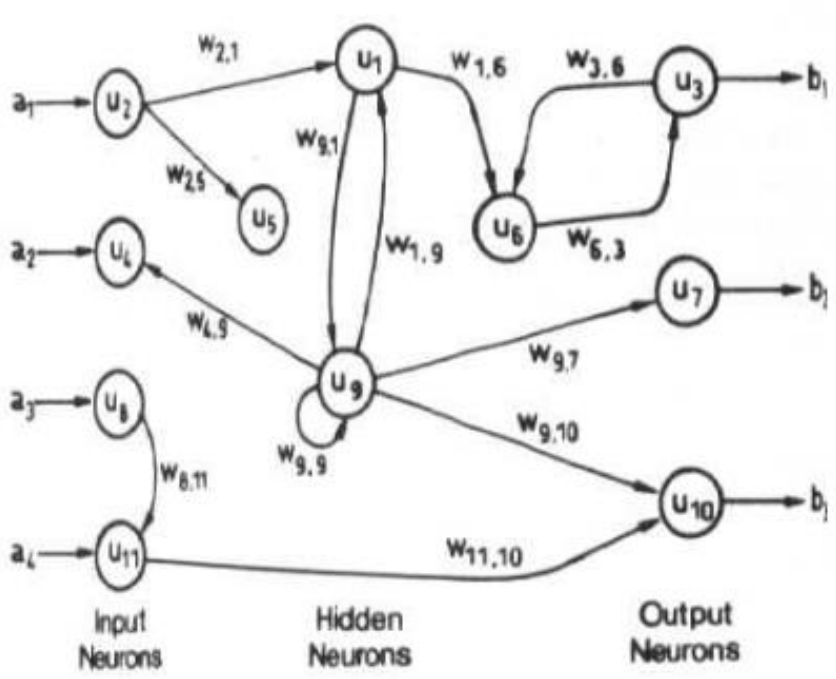


Рисунок 2.2 – Загальний вигляд нейронної мережі

Тут мережа приймає на вхід вектор (a1, a2, a3, a4) і видає результат (b1, b2, b3). Звичайно структура нейронної мережі являє собою послідовність з'єднаних між собою шарів, причому для кожної пари сусідніх шарів задані ваги wi,j між i -м нейроном 1 шару і j-м нейроном 2-го (при цьому не обов'язково з'єднувати всі пари нейронів). Таким чином, якщо u (k) i- вихід i-го нейрона k-го шару, то на вхід Кj-му нейрону k + 1-го шару подається $\sum_i w_{i,j}^{(k)} \cdot u_i^{(k)}$.

Всі шари крім першого (вхідного) і останнього (вихідного) називаються прихованими. Вихід нейрона визначається його активаційною функцією, яка застосовується до входу. Популярними функціями є:

а) сигмоїда σ (z) = 1/1 + e^z;

б) порогова φ (z) = max (0, z);

в) гіперболічний тангенс φ (z) = 2/1 + e^{-2z};

г) тотожня функція φ (z) = z.

Таким чином, в базовому випадку мережа виглядає, як показано на рисунку 2.3:

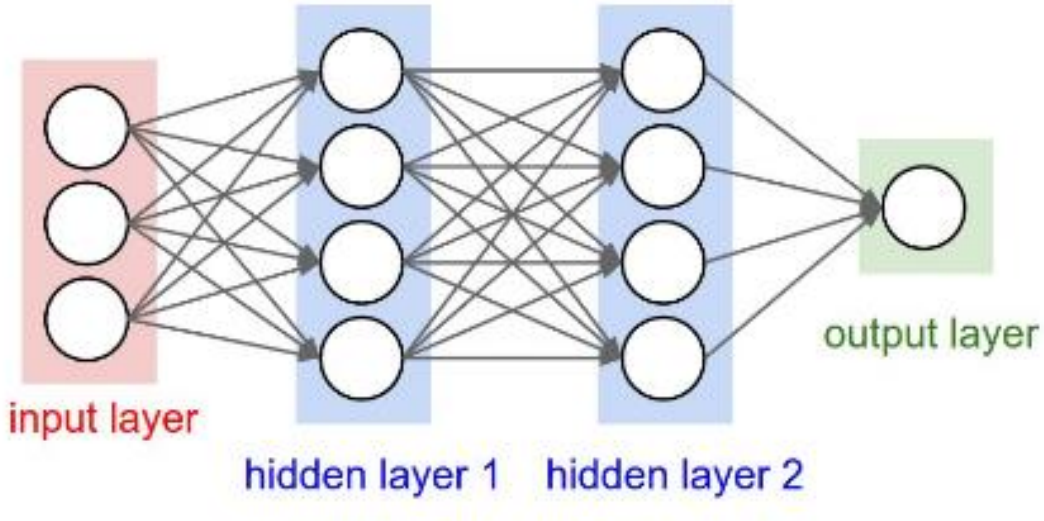


Рисунок 2.3 – Нейронна мережа в базовому випадку

Подібні шари називаються лінійними, оскільки являють собою добуток вхідного вектора на матрицю ваг w(k)i,j на k-му шарі. На практиці такому перетворенню зазвичай додають зміщення bk, тобто виходний вектор k-го шару u(k) обчислюється через попередній шар як:

$$u^{(k)} = \phi^{(k)}\left(A^{(k)} \cdot u^{(k-1)} + b_k\right), A^{(k)} \in R^{d_1 \times d_2}, b_k \in R^{d_1} \quad (2.5)$$

У разі диференційованих функцій активації перетворення f (x), що задається мережею також диференційоване, що дозволяє обчислювати градиєнти в точці по параметрам w (k) i, j.

При умові задання навчальної вибірки {(xi, yi)} і функціоналу втрат L (yi, f (xi)) (також дифференційованого) стає можливим оптимізувати параметри мережі за допомогою методу градієнтного спуску. Такий спосіб навчання отримав назву методу зворотного поширення помилки оскільки обчислення градієнта йде в протилежному порядку по порівнянню з обчисленням значення функції f (x).

Також існують спеціальні нелінійні шари, заточені під рішення різних класів задач. Наприклад, згорткові шари, що приймають на вхід зображення (зазвичай двомірні матриці) і на виході отримують комбінації їх згорток з різними ядрами. Також існують перетворення, які здійснюють згортки з функцією max або min - max і min-pooling відповідно. Подібні підходи в даний момент рахуються найбільш використовані в аналізі зображень. Крім того, є шари, що допомагають з проблемою перенавчання - dropout (прибирає частку зв'язків між шарами для занадто хорошої підгонки під тренувальні дані) або batchnormalization (нормалізація даних не тільки перед завантаженням їх в мережу, але й у процесі проходження між шарами). Всі згадані підходи також дозволяють рахувати градієнти по параметрам мережі та відповідно навчати модель методом градієнтного спуску.

Однак в розглянутій задачі зіставлення відеозапису послідовності векторів ознак Y = y1,. . . , YT може бути досить довгою, T не завжди фіксоване. Також потрібно на виході отримувати послідовність слів, а не єдину мітку класу. У зв'язку з подібними проблемами з'явилися рекурентні нейронні мережі, які мають "зворотній зв'язок", іншими словами дозволяють передавати накопичену інформацію далі в часі.

Візуально такий процес можна зобразити, як показано на рисунку 2.4 :

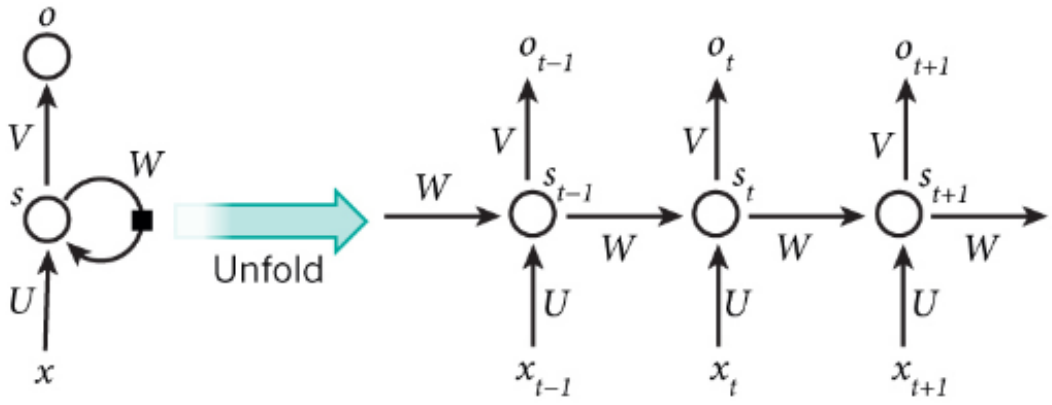


Рисунок 2.4 – процес рекурентної нейронної мережі

Тут мережа в момент часу т приймає вхідний вектор xt, прихований стан на попередньому кроці st-1 і обчислює вихідний вектор ot. Після цього новий

стан st передається в наступну ітерацію процесу. Такі мережі дозволяють обробляти послідовності невідомої довжини, враховуючи зв'язки між теперішнім і минулим.

Основним методом навчання таких мереж є метод зворотного поширення помилки в часі. Він працює таким чином: здійснюється "розгортка" мережі в часі, тобто якщо вхідна послідовність мала довжину T, то мережа скопіюється T разів. Після цього мережа розглядається як не рекурентна, тому можна запустити звичайний алгоритм зворотнього поширення помилки і порахувати градієнти за вагами (існують варіанти запускати перерахунок ваг відразу для всієї послідовності довжини T або для кожного префікса довжини $1 \leq T' \leq T$). Потім мережа назад згортається в початковий стан і відбувається оновлення ваг середнім значенням градієнта з усіх копій мережі.

Найпростіші архітектури подібних мереж хоч і здатні обробляти послідовності довільної довжини, але схильні до таких проблем при навчанні як "проблема зникнення градієнта" і "проблема вибуху градієнта". Вони пов'язані з тим, що при послідовному збільшенні маленьких / великих чисел результат зростає / падає експоненціально і градієнт при зворотному поширенні помилки може стати дуже великим або дуже близьким до нуля. У звичайних нейронних мережах така проблема також присутня, але в рекурентних моделях вона виражена особливо гостро. З тих же причин, такі мережі погано враховують залежності між порівняно далекими у часі моментами.

Для боротьби із згаданими проблемами була придумана спеціальна архітектура під назвою Long Short Term Memory . Візуально шар такої мережі можна зобразити, як показано на рисунку 2.5:

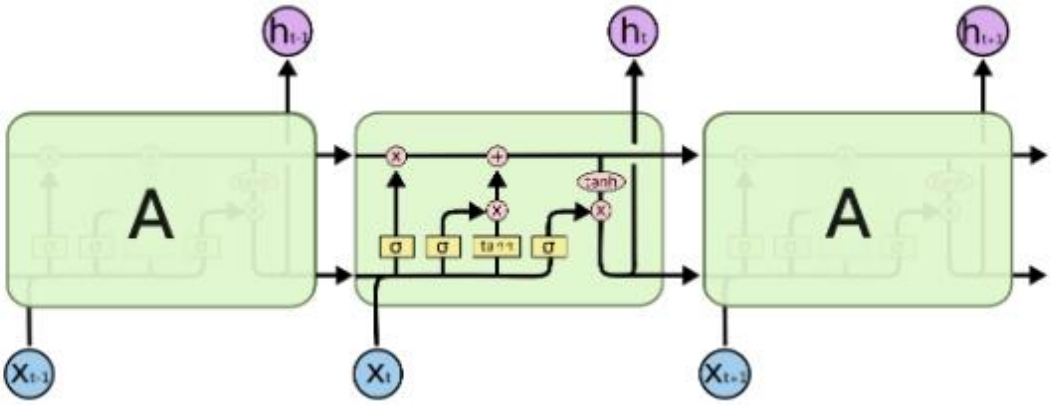


Рисунок 2.5 – LSTM загальний випадок

Головна ідея такої мережі - це введення так званих cell state, до якого в момент часу t послідовно застосовуються forget gate і update gate. Таким чином стан мережі проходить крізь такі перетворення видаливши частину інформації, додавши нову і якусь частку старого стану залишивши майже без зміни. Таким чином довгострокові залежності можуть довго зберігатися при роботі такої мережі. Крім того через нормалізацію даних за допомогою сігмоїдної і тангенсної активаційних функцій проблеми зникаючого і вибухаючого градієнта набагато менше проявляють себе. Подібна архітектура дуже часто використовується при роботі з рекурентними нейронними мережами і переважній більшості випадків виходять моделі кращої якості, ніж при використанні стандартних рекурентних моделей.

Рекурентні нейронні мережі успішно використовуються при побудові акустичних і язикових моделей.

Одним з недоліків рекурентних мереж є те, що стан в конкретній одиниці часу має знання тільки про минулі входи до певної точки у реченні, але не має знання про майбутні стани. У деяких додатках, таких як мовне моделювання, результати значно покращилися із знаннями про минулі і майбутні стани. Конкретним прикладом є розпізнавання рукописного тексту, в якому є явна перевага використання знань про обидва

минулого і майбутніх станів, оскільки він забезпечує краще уявлення про контекст, що лежить в основі.

У двонаправленій рекурентній мережі ми маємо окремі приховані стани h t і h t для напрямків вперед і назад. Прямі приховані стани взаємодіють тільки з кожним іншим і те ж саме стосується зворотних прихованих станів. Головна відмінність полягає в тому, що форвардні стани взаємодіють у напрямку вперед, у той час як зворотні стани взаємодіють назад. Обидва h t і h t, однак, отримують вхід від одного і того ж вектора x t і вони взаємодіють з одним і тим же вихідним вектором t. Загалом, будь-яку властивість поточного слова можна передбачити більш ефективно, використовуючи цей підхід, тому що він використовує контекст з обох сторін. Наприклад, впорядкування слів в декількох мовах дещо різні в залежності від граматичної структури.

Gated Recurrent Unit (GRU) є типом рекурентних нейронних мереж (RNN), які покращують попередні RNN, додаючи «cells and gates» для розповсюдження інформації про більшому проміжку часу і навчання керувати цим інформаційним потоком. Він подібний до LSTM RNN.

$$[u_t, r_t]^\top = sigm\big(W_z z_t + W_h h_{t-1} + b_g\big)$$
$$\tilde{h} = \tanh(U_z z_t + U_h(r_t \odot h_{t-1}) + b_h)$$
$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t \qquad (2.6)$$

де z: = {z1, ..., zT} є вхідною послідовністю до RNN;

Θ позначає поелементне-множення;

asigm (r) = 1 / (1 + exp (-r)).

Двонаправлений GRU (Bi-GRU), був введений в контексті LSTM:

одна рекурентна мережа діє $\{z_T,...,z_1\} \rightarrow \{h_1,...,h_T\}$, а інша навпаки.

Bi-GRU гарантує, що ht залежить від zt′ для всіх t′. Щоб параметризувати розподіл послідовностей в момент часу t покладемо:

p(ut|z) = $\mathrm{softmax}(\mathrm{mlp}(\mathbf{h}_t; \mathbf{W}_{mlp}))$, де mlp є мережею прямого доступу з вагами Wmlp.

Тоді ми можемо ввести розподіл довжиною T як:

$$p(u_1, \ldots, u_T|\mathbf{z}) = \Pi_{1 \le t \le T} p(u_t|\mathbf{z}), \quad (2.7)$$

де Ti визначається за допомогою z(вхідні дані до GRU). Тобто в нашій моделі, z є виходом STCNN.

Згорткові нейромережі:

Згорткові мережі, також відомі як згорткові нейронні мережі або CNNs, є спеціалізованим видом нейронної мережі для обробки даних що має відому топологію, подібну до сітки. Приклади включають дані часових рядів, які можуть розглядатися як 1D сітка, що приймає зразки через регулярні інтервали часу, і дані зображення, які можна розглядати як 2D-сітку пікселів. Такі мережі виявились надзвичайно успішним у практичному застосуванні. Назва вказує, що мережа використовує математичну операцію, яку називають згортка. Згортка - це спеціалізований вид лінійної операції. Згорткові мережі - це просто нейронні мережі, які використовують згортку замість загальної матриці розмноження принаймні в одному з їх шарів.

Згорткові нейронні мережі (CNNs) відіграли важливу роль у підвищенні продуктивності в задачах комп'ютерного зору, таких як розпізнавання об'єктів. Базова згортка:

$$[\text{conv}\,(\text{x}, \text{w})]_{c\prime ij} = \sum_{c=1}^{C} \sum_{i'=1}^{k_w} \sum_{j'=1}^{k_h} w_{c\prime ci\prime j\prime}\, x_{c,i+i',j+j\prime}, \quad (2.8)$$

де x-вхідні значення, $R^{C` x\, C\, x\, k_w\, x\, k_h}$

xcij= 0 для i,j поза межами.

Типи шарів згорткової нейронної мережі:

Існує багато типів шарів, які використовуються для побудови згорткової нейронної мережі :

а) Згортковий (CONV)

б) Активація (RELU)

в) Об'єднання (POOL)

г) Повнозв’язний (FC)

д) Нормалізація (BN)

э) Відключення (DO)

Укладання ряду цих шарів специфічним чином дає CNN. Наприклад, типовий вид CNN: INPUT => CONV => RELU => FC => SOFTMAX
Тут ми визначаємо простий CNN, який приймає вхід, застосовує шар згортки, потім активаційний шар, потім повнозв’язний шар, і, нарешті, класифікатор softmax для отримання виходу ймовірності класифікації. Шар активації SOFTMAX часто опускається з мережевої діаграми, як передбачається, він безпосередньо слідує за кінцевим FC.

З цих типів шарів, CONV і FC, (і в меншій мірі, BN) є єдиними шарами параметрів, які вивчаються під час тренувального процесу. Активаційні та відсічні шари не вважаються справжніми «шарами» самі по собі, але часто включаються в мережеві діаграми для створення явно зрозумілої архітектури. Базові шари (POOL), які мають рівне значення як CONV і FC, також є включені в мережеві діаграми, оскільки вони мають суттєвий вплив на просторові розміри зображення.
CONV, POOL, RELU і FC є найбільш важливими при визначенні фактичної архітектури.

Згорткові шари

Шар CONV є основним будівельним блоком згорткової нейромережі . Параметри шару складаються з набору K-фільтрів, що вивчаються ("ядро"), де кожен фільтр має ширину та висоту, і майже завжди квадратний. Ці фільтри невеликі (з точки зору їх просторових розмірів) але простягаються по всій глибині обсягу.

Для входів в CNN глибина - це кількість каналів на зображенні (тобто глибина три при роботі з RGB зображеннями, по одному для кожного каналу).

Після застосування всіх K-фільтрів до вхідного об'єму ми маємо K 2-мірних карти активації. Потім складаємо наші K активаційних карти вздовж глибинного розміру нашого масиву, щоб сформувати кінцевий вихід.

Кожен вихід, таким чином, є виходом нейрона, який є лише малою областю. Таким чином, мережа «вивчає» фільтри, які активуються, коли вони бачать конкретний тип функції у заданому просторовому розташуванні у вхідному обсязі. У нижніх шарах мережі фільтри може активуватися, коли вони бачать регіони, подібні до країв або кутів. Потім у більш глибоких шарах мережі фільтри можуть активуватися при наявності високого рівня особливості, такі як частини обличчя, лапа собаки, капот автомобіля і т.д.

Функції активації

Після кожного шару CONV в CNN ми застосовуємо нелінійну функцію активації, таку як ReLU. Зазвичай ми позначаємо активаційні шари як RELU в мережевих діаграмах, оскільки, активації ReLU найчастіше використовуються,

Активаційні шари не є технічно «шарами» (через те, що немає параметрів / ваг) і іноді пропускаються з діаграм архітектури мережі. Передбачається, що активація безпосередньо слідує за згорткою. Як приклад розглянемо наступну мережу: INPUT => CONV => RELU => FC.

Щоб зробити цю діаграму більш лаконічною, ми можемо просто видалити компонент RELU, оскільки передбачається, що активація завжди слідує за згорткою: INPUT => CONV => FC.

Шар активації приймає вхідний обсяг вхідного сигналу розміру Wвхід × H вхід × D вхід і потім застосовується дана функція активації . Оскільки функція активації застосовується в поелементно, таким чином, вихідний шар активації завжди є таким же, як вхідний розмір, W вхід = W виведений, H вхідний = H виведений, D вхідний = D виведений.

Об'єднання шарів

Існує два способи зменшення розміру вхідного обсягу - шари CONV з кроком> 1 (який ми розглянули) і шари POOL. Загальноприйнято вставляти шари POOL між послідовними шарами CONV в архітектурах CNN:
INPUT => CONV => RELU => POOL => CONV => RELU => POOL => FC

Основною функцією шару POOL є поступове зменшення просторового розміру (тобто ширини і висоти) вхідного обсягу. Це дозволяє зменшити кількість параметрів і полегшує обчислення в мережі - об'єднання також допомагає контролювати перенавчання.

Рівні POOL функціонують на кожному з глибинних фрагментів входу незалежно, використовуючи або max або функцію середнього. Максимальне об'єднання зазвичай виконується в середині архітектури CNN для зменшення просторового розміру, тоді як середній пул зазвичай використовується як кінцевий шар мережі. Найбільш поширеним типом шару POOL є max pooling, хоча ця тенденція змінюється з введенням більш екзотичних мікроархітектур.

Зазвичай використовується розмір пулу 2 × 2, хоча більш глибокі CNN, які використовують великі вхідні зображення (> 200 пікселів) можуть використовувати розмір пулу 3 × 3 на початку архітектури мережі. Ми також

Повнозв’язний шари

FC -шари завжди розміщуються в кінці мережі (тобто, ми не застосовуємо шар CONV, а потім шар FC, за яким слідує інший CONV шар). Загальноприйнято використовувати один або два шари FC перед застосуванням класифікатора softmax:
INPUT => CONV => RELU => POOL => CONV => RELU => POOL => FC => FC

Тут ми застосовуємо два повністю пов'язаних шари перед класифікатором softmax, який буде обчислиювати кінцеві ймовірності виходу для кожного класу.

Просторово-часові згорткові нейронні мережі (STCNNs) можуть обробляти відеодані за допомогою згорток в часі. Звідси аналогічно:

$$[\text{stconv}(x, w)]_{c'tij} = \sum_{c=1}^{C} \sum_{t'=1}^{k_t} \sum_{i'=1}^{k_w} \sum_{j'=1}^{k_h} w_{c'ct'i'j'} x_{c,t+t',i+i',j+j'} \quad (2.9)$$

Сучасні додатки із застосуванням згоркових нейронних мереж часто включають мережі, що містять більше одного мільйона одиниць. Реалізації, що використовують паралельні обчислення , є надто важливими . Однак у багатьох випадках, також можна прискорити згортку, вибравши відповідний алгоритм згортки.

Втрата тимчасової класифікації з'єднання (CTC) широко використовується у розпізнаванні мови, оскільки усуває необхідність підготовки даних.

Виходом даної моделі є послідовність з дискретним розподілом по класам маркерів (класи задані в словнику), доповнених спеціальним «пустим» маркером. CTC обчислює ймовірність послідовності шляхом маргіналізації над усіма послідовностями, які визначені як еквівалентні цій послідовності. Нехай V позначає множину токенів, які модель класифікує за один крок свого виходу (словник), а порожній доповнений словник V~ = V∪ {_}}

де _ позначає порожній символ CTC

Підхід CTC є методикою для маркування послідовності, використовуючи RNN, де вирівнювання між входами і цільовими мітками невідомі. CTC може бути реалізований з вихідним шаром softmax, що використовується для оцінки ймовірності вихідної мітки на даний момент. Вірогідність вихідних міток з мережі визначають розподіл ймовірностей по всіх можливих мітках вхідних послідовностей, включаючи порожні мітки. Мережу можна навчити оптимізувати загальну ймовірність правильних міток для тренувальних даних, оцінених за допомогою мережевих виходів.

Правильні позначення для послідовності виходів визначаються як сукупність всіх можливих міток введення з цільовими мітками в правильній послідовності, можливо з повторами і з порожніми мітками, дозволеними між окремими мітками.

Хоча звичайні системи розпізнавання мови та розпізнавання рукописного тексту, як правило, навчаються з фіксованих вирівнювань, використання алгоритму прямого зворотного зв'язку для відновлення цільових мереж завдяки поточній моделі може однаково застосовуватися до звичайних мереж або рекурентних .

Отже, CTC відрізняється від звичайних моделей двума основними критеріями. По-перше, додатковий маркер звільняє мережу від створення прогнозування міток у кадрі, коли це не потрібно. По-друге, критерій навчання оптимізує ймовірність станів, а не ймовірність вхідних даних.

Огляд методів виділення ознак.

Серед підходів до виділення ознак з об'єктів на зображенні можно виділити алгоритми, засновані на виділенні контурів об'єкта (в даному випадку - губ) і методи, що працюють безпосередньо зі значеннями пікселів в деякій області на зображенні. Також існують алгоритми, що використовують методи трекінгу особи для поліпшення точності ознак в разі, якщо входом є не нерухоме зображення, а відеоряд .

Дескриптори, засновані на виділенні контурів.

До класичних методів першої групи можна віднести активні моделі зовнішнього вигляду (active appearance models) і активні моделі форми (active shape models) . Вони являють собою методи підгону деякої статистичної моделі зображень, що враховує форму або текстуру об'єкта, під конкретну картинку. На схожих ідеях засновано безліч підходів до знаходження ключових точок на зображеннях. Виходом даних алгоритмів є безліч ключових точок на

зображенні, що позначають форму особи, губ, очей і т.д. Існують готові реалізації деяких підходів розмітки особи, описаних в статтях. Наприклад, на зображення прикладу роботи програми IntraFace показано на рисунку 2.6 :

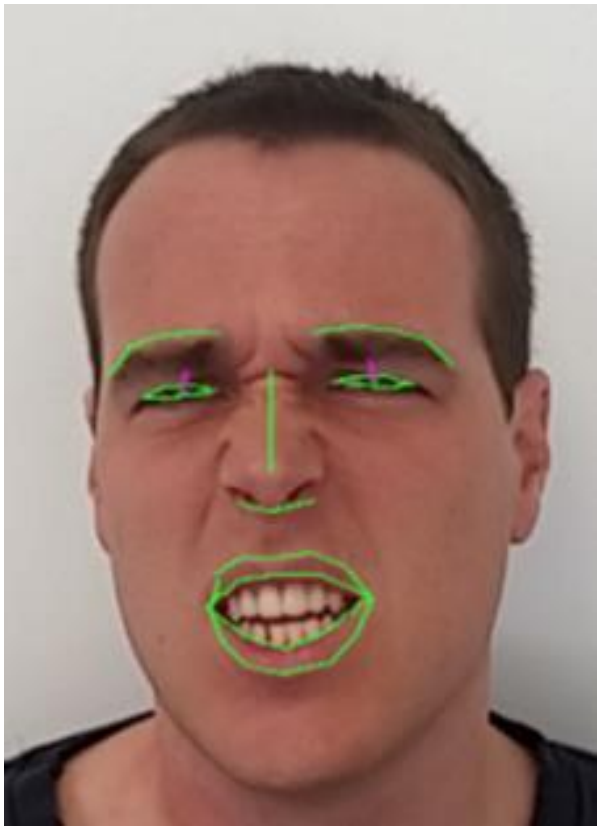


Рисунок 2.6 – приклад розмітки обличчя за допомогою InterFace

Також широко відомий метод, що застосовано в даній роботі. Він заснований на підгоні статистичної моделі форми особи до заданого зображення за допомогою градієнтного бустінгу на регресійних деревах.

На рисунку 2.7 наведені приклади роботи, отримані з використанням бібліотеки dlib, де даний підхід реалізований (показані тільки контури губ):

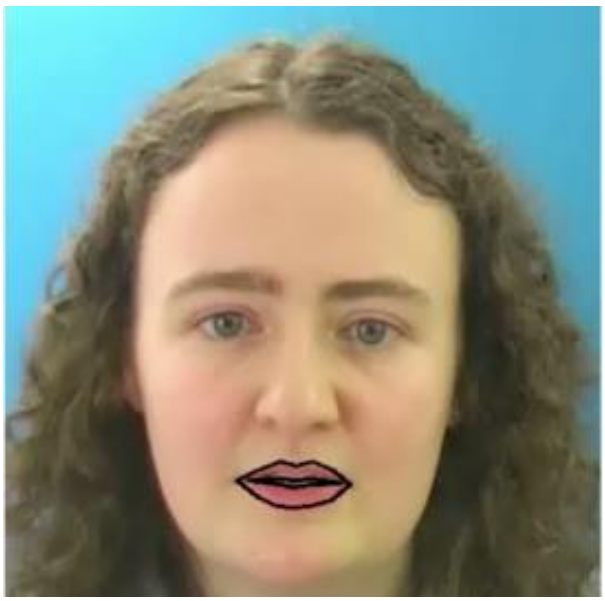


Рисунок 2.7 – приклад розмітки губ за допомогою dlib

У даній роботі в якості базового алгоритму виділення ознак був використаний останній згаданий підхід, оскільки за якістю його можна порівняти з кращими рішеннями і при цьому він має реалізацію з відкритим кодом, що може дозволити в майбутньому переробляти його під потрібні завдання.

Дескриптори, засновані на аналізі значень пікселів.

До другої групи можна віднести такі підходи як навчання вилучення вектора ознак з набору пікселів зображення за допомогою нейронних мереж. Такі мережі можуть працювати таким способом: на вхід подаються значення пікселів зображення або регіону інтересу і на виході очікується вектор, в точності рівний вхідному, при цьому приховані шари містять набагато менше нейронів, ніж вхідний. У процесі навчання відбувається оновлення ваг прихованих шарів, щоб вони видавали компактне представлення вихідних даних, з якого можна з певною мірою точності їх відновити. Перевагою такого методу є відсутність необхідності вручну вирішувати які ознаки найкраще опишуть вхідні дані для конкретного завдання, придумувати різні ручні дескриптори. Також такий підхід менше залежить від способу подачі даних,

наприклад, можна обчислювати ознаки відразу для декількох поспіль зображень з відео-потоку.

Також варто згадати про алгоритми засновані на методі головних компонент. Він може використовуватися для стиснення вектора ознак великої розмірності (наприклад, списку значень пікселів регіону інтересу). У його основі лежить застосування ортогонального перетворення до вихідного простору таким чином, щоб вибіркова дисперсія уздовж першої координати була максимальна і далі вибіркова дисперсія уздовж кожної наступної координати максимальна, за умови її ортогональності всім попереднім координатами. Для застосування подібних підходів на практиці потрібно попередньо обчислити матрицю ортогонального перетворення і вектор середніх по навчальній вибірці. Після цього робота з заданим тестовим зображенням являє собою віднімання середнього вектора з подальшим перетворенням в вектор маленької розмірності, який можна розглядати як вектор ознак і використовувати для подальшої класифікації як компактне представлення вихідного зображення.

2.2 Критерії якості роботи системи

Перед тим, як робити аналіз якості розпізнавання слів, фраз, або речень, згідно постановці задачі, відеозапис повинен пройти деякі попередні етапи підготовки, що полягають в обробці відеозапису: вилучення потрібних характеристик, обчислення та аналіз характеристик попередніх кадрів на відеозаписі, виявлення крайніх кадрів фрази, та виділення остаточної фрази або команди.

а) Площа, що займають в кадрі обличчя, та губи.

Для того, щоб алгоритм працював належним чином, необхідно якомога краще виділити ті елементи, з якими буде проведена робота надалі. В даному випадку це зона губ, та роту. В кадрі обличчя повинно бути оптимального

розміру, що означає, воно не повинно бути присутнім у кадрі лише частково, та не повинно бути занадто далеко, тому що в таких випадках програмний продукт просто не зможе розрізнити обличчя. Після кроку, коли обличчя, та необхідні точки на ньому (точки, що відповідають зоні губ, та роту) виявлено, обраховується площа, яку займають губи в кадрі. Таким чином, аналізуючи даний показник, можна робити деякий аналіз відозаписів. Приклад розмітки губ, по точкам якої рахується площа, показано на рисунку 2.8 :

Рисунок 2.8 – Приклад розмітки губ

б) Дисперсія площі, порахованої в попередньому пункті, обрахована за попередні N кадрів.

Після вилучення площі на кожному кадрі відеозапису, можна аналізувати даний показник зі зміною в часі. В даній роботі використовується аналіз міри

відхилення значень площі, від середнього значення. Але, так як обробка застосовується в реальному часі, то даний показник на кожному кроці обраховується відносно минулих N кадрів. Тобто, змінюючи параметр N, можливо корегувати “історію”. Якщо N задати великим, то аналіз буде більш загальним, та складніше буде працювати з конкретним кадром в даний момент часу. Але якщо N задати занадто малим, то вибірка буде занадто малою, а це означає, що ми не будемо мати достатньо інформації про зміну показника в часі. Таким чином, аналізуючи дисперсію, можна зробити висновок про стан спікера в конкретний момент часу. У випадку, коли дисперсія велика, та продовжує зростати, вірогідно, спікер активно говорить. Якщо щ даний показник малий, та продовжує зменшуватись, можна сказати що в даний момент часу спікер зробив паузу, тобто мовчить.

в) Довжина фрази.

Кожна людина має свою швидкість, з якою вимовляються слова. Тому одна й та сама фраза чи речення можуть бути сказані у різний проміжок часу, тобто відповідно, одна й та сама фраза буде займати різну довжину кадрів. У наборі даних, на якому проводилися дослідження, одна фраза триває 3 секунди. Так як частота кадрів в секунду дорівнює 25, то для даного набору даних було вирішено обрати параметр мінімальної довжини фрази 50 кадрів. Але, зрозуміло, якщо будуть змінюватись характеристики відеозапису, такі як частота кадрів, але якщо спікер буде говорити занадто повільно, чи занадто швидко, то відповідний параметр потрібно буде змінювати. Обмеження по довжині фрази було введено для таких випадків, коли людина мовчить , але показник дисперсії для минулих кадрів знаходиться на межі двох станів (руху, та спокою). В такому випадку, необхідно розуміти, в якому випадку людина дійсно мовчить, а я якому розпочинається фраза. Тому за допомогою коефіцієнту, що характеризує довжину мінімальної можливої фрази, відсіюються випадки, в яких помилково детектується фраза довжиною в декілька кадрів, чого не може бути фізично.

2.3 Алгоритм

Відео обробляється детектором обличчя DLib (Sagonas et al., 2013) та предиктором точок обличчя з 68 орієнтирами, у поєднанні з фільтром Калмана. Використовуючи ці орієнтири, застосовуються афінне перетворення, щоб отримати зображення роту розміром 100 × 50 пікселів на кадр.

Після отримання точок, що відповідають позиції роту, обраховується площа, що займає рот, як площа многокутника по даним координатам:

$$A = \frac{1}{2}\left|\sum_{i=1}^{n} x_i(y_{i+1} - y_{i-1})\right| = \frac{1}{2}\left|\sum_{i=1}^{n} y_i(x_{i+1} - x_{i-1})\right| =$$
$$= \frac{1}{2}\left|\sum_{i=1}^{n} x_i y_{i+1} - x_{i+1}y_i\right| = \frac{1}{2}\left|\sum_{i=1}^{n} det\begin{pmatrix} x_1 & y_i \\ x_{i+1} & y_{i+1} \end{pmatrix}\right| \qquad (2.10)$$

Знаючи площу, що займає рот на кожному кадрі, ми можемо порахувати дисперсію вибірки для даного показника:

$$\sigma_y^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 = \left(\frac{1}{n}\sum_{i=1}^{n} y_i^2\right) - \bar{y}^2 = \frac{1}{n^2}\sum_{i<j}(y_i - y_j)^2 \qquad (2.11)$$

Тобто ми отримуємо відхилення площі роту від середнього значення. Дисперсія рахується для кожного кадру, відносно попередніх 20 кадрів. Тобто якщо даний показник буде малим, це означає що людина знаходиться у стані спокою. Якщо ж цей показник є великим, то людина розмовляє. Таким чином, з'являється можливість розділити команди в режимі реального часу. В даній роботі вважається, що одна команда має бути не менше ніж 50 кадрів.

Структура моделі:

Після того, як алгоритм детектував паузу, здійснюється відправка частини відео, де була сказана команда на алгоритм розпізнавання команди.

Він, в свою чергу, робить передбачення сказаної фрази, отримуємо як результат текстову команду, та продовжуємо обраховувати дисперсію для наступних кадрів, щоб відслідкувати наступну команду.

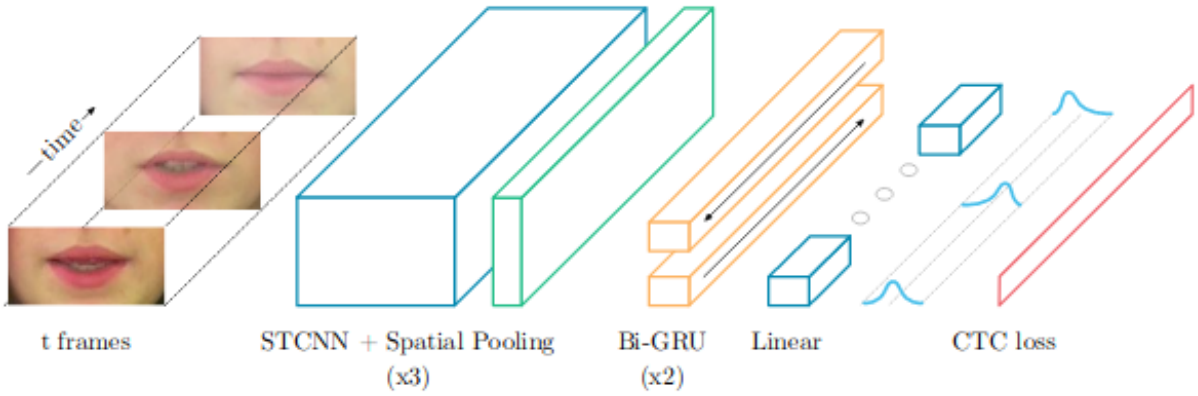


Рисунок 2.13 – Схема моделі розпізнавання

Модель містить три згорткові шари, після цього вихідні дані цих шарів подаються на два рекурентні шари. Нарешті, застосовується лінійне перетворення для кожного кадру із функцією softmax та використовуючи CTC отримуємо вихідні дані як строку. Всі шари використовують функцію активації ReLu.

На високому рівні архітектури, кадри вилучені з відеопослідовності, обробляються в малих наборах за допомогою CNN, тоді як LSTM працює на виході CNN послідовно для створення вихідних символів. Точніше, послідовність з 10 кадрами групується разом у блоці (ширина x висота x 10), довжина послідовності може змінюватися, але послідовний характер цих кадрів створює просторово-часовий CNN.

Потім вихід LSTM, що називається Gated Recurrent Unit (GRU), обробляється багатошаровим персептроном (MLP), отримуючи вихідні значення для різних символів, отриманих з CNN. Нарешті, CTC забезпечує

остаточну обробку результатів послідовності, щоб зробити її більш зрозумілою з точки зору точних виходів, тобто слів і речень. Такий підхід дозволяє передавати інформацію через періоди часу, що містять як слова, так і, в кінцевому рахунку, речення, підвищуючи точність прогнозів мережі.

Модель також використовує додатковий алгоритм, який зазвичай використовується в системах розпізнавання мовлення - (CTC). Після класифікації символів кадру, які в поєднанні з більшою кількістю символів визначають вихідну послідовність, CTC може згрупувати ймовірності декількох послідовностей (наприклад, "к__іі_тт" і "ккіііі__т") в одне і те ж слово (у цьому випадку "кіт") остаточний прогноз.

Прогнозуючи символи алфавіту та додатковий символ "_" , можна створити прогноз слова, видаляючи повторювані букви і порожні символи. Практично це означає, що витягнуті вимови, варіації акценту і таймінгів, а також паузи між складами і словами можуть бути більш прогнозованими, використовуючи CTC для виходів.

2.4 Висновки за розділом

У розділі було проаналізовано математичні обгрунтування методів розпізнавання мови, використовуючи лише відеозображення. Проаналізувавши підходи, що використовувалися в цій сфері, було обрано структуру моделі розпізнавання, та розроблено алгоритм поділу суцільного відеопотоку на окремі фрази.

Також, описано новий підхід до вилучення ознак області рота, такі як площа , середньо квадратичне відхилення, та довжина сказаної фрази. У порівнянні з попередніми методами, це рішення має переваги , за рахунок яких можливо робити одночасний процес обробки зображень у реальному часі.

РОЗДІЛ 3 АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Вибір платформи

В процесі аналізу та дослідження існуючої проблеми, було прийнято рішення використовувати нейронні мережі, а також деякі математичні підходи, зокрема математичну статистику, для розробки програмного продукту.

В програмному рішенні використовується аналіз математичних величин, таких як математичне сподівання та дисперсія — для вдалого розділення відеопотоку на частини (фрази), а також згорткові, та рекурентні нейронні мережі — для обробки отриманих частин.

Оскільки сьогодні існує багато доступних мов програмування, звичайно першим питанням постає вибір мови . Для розробки була обрана мова програмування Python. Оскільки вона вважається зручною для роботи з:

а) нейронними мережами

б) математичною статистикою

в) зображеннями та відеопотоками

г) паралельними обчисленнями

Мова програмування Python має такі характеристики:

а) розумілість

Python є дуже читабельним, код призначений бути зручним для читання і, отже, підтримується набагато більше, ніж традиційні мови сценаріїв. Код Python легко зрозуміти, навіть якщо ви його не написали. Крім того, Python має глибоку підтримку механізмів програмного забезпечення , такі як об'єктно-орієнтованість (ОО) та функціональне програмування.

б) Продуктивність

Python підвищує продуктивність розробників у багато разів , тому що він більш компактний, тобто його менше набирати, менше налагоджувати і

менше підтримувати. Програми Python також одразу, без тривалих кроків компіляції.

в) Переносимість програми

Більшість програм Python працює на всіх основних комп'ютерних платформах . Перенесення, наприклад, коду Python між Linux і Windows, як правило, є лише питанням копіювання коду сценарію між машинами. Крім того, Python пропонує кілька опцій для розробки графічних інтерфейсів користувача, програм доступу до баз даних, веб-систем.

г) Підтримка бібліотек

Python поставляється з великою колекцією готових функцій та бібліотек, наприклад для обробки зображень, відео, та розробка нейронних мереж.

3.2 Аналіз вимог користувача до програмного продукту

Проаналізувавши вимоги до програмного продукту, зроблено висновок, що користувач повинен отримати декілька режимів роботи програми, зображних на рисунку 3.1, а саме:

а) обробка відеопотоку в реальному часі

б) обробка існуючого відеозапису

в) аналіз математичних показників (зображення залежності площі , що займають губи, від часу)

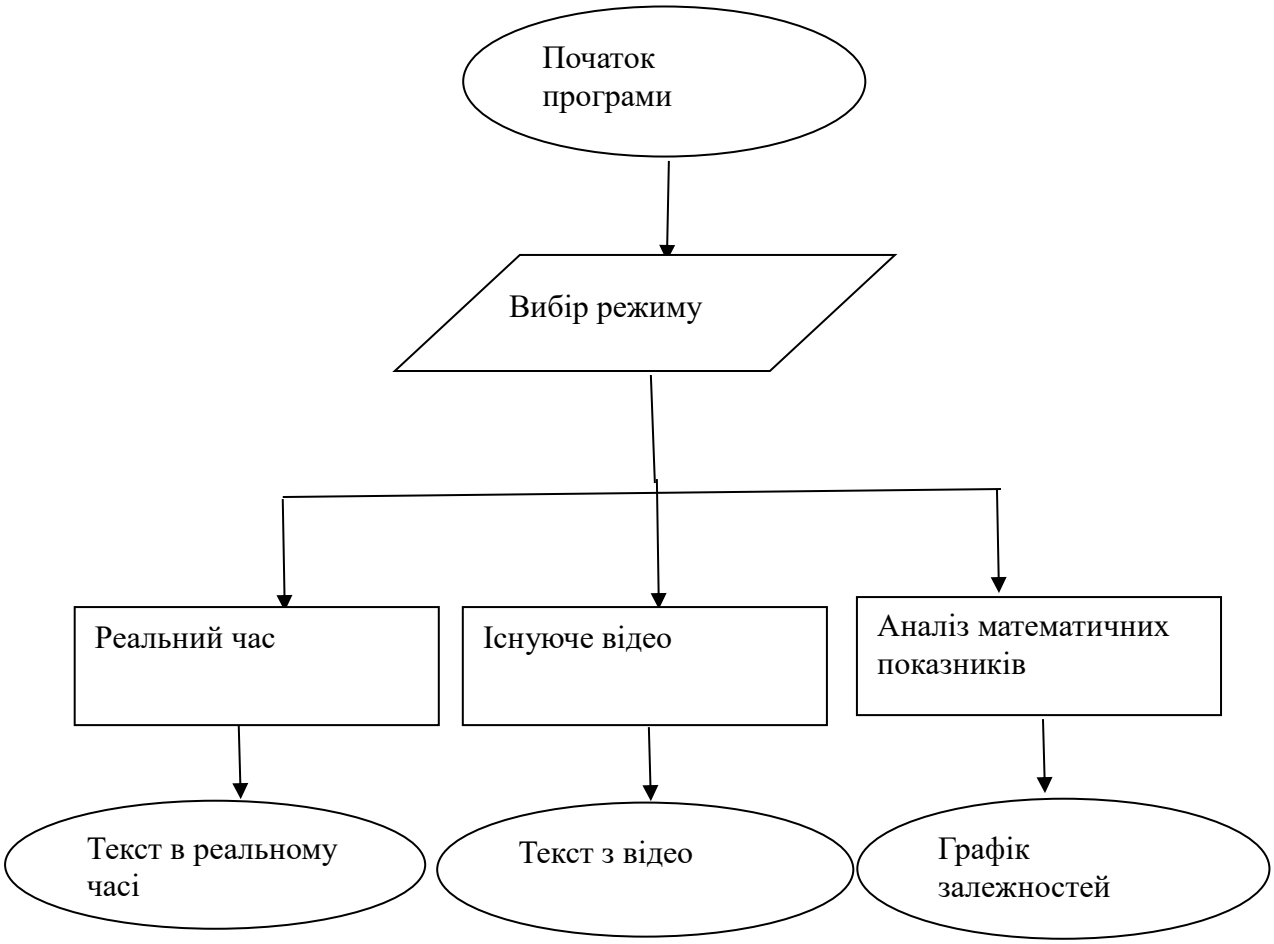


Рисунок 3.1 – Режими роботи програми

3.3 Аналіз архітектури програмного продукту

Проаналізуємо архітектуру розробленого програмного продукту. Після запуску програми, користувачу пропонується обрати один з трьох режимів:

а) режим реального часу

б) режим обробки існуючого відеозапису

в) зображення залежностей математичних величин

Розглянемо кожен з цих режимів окремо:

Режим реального часу

Даний режим являє собою реалізацію обробки всіх характеристик, метрик, та показників у реальному часі. Для такої задачі надто важлива швидкість обробки та розрахунку показників. Тому було прийнято рішення розробити даний алгоритм за допомогою паралельних обчислень. На відміну від звичайного режиму, в якому після отримання зображення з відеопотоку, одразу обчислюються характеристики, задачі будуть розділені на два потоки:

а) зчитування кадрів з камери

б) обробка отриманих кадрів

Тобто в першому потоці здійснюється обробка відеокамери, та послідовне збереження кожного кадру. А в другому потоці виконуються всі обчислення, пов’язані з характеристиками відео, детектуванням обличчя та губ, розрахунок математичних показників. Таким чином, ми забезпечуємо необхідну для розпізнавання частоту кадрів.

Так як обчислення, що проводяться в другому потоці мають набагато більшу обчислювальну складність , ніж зчитування кадрів в першому потоці, то, зрозуміло, що алгоритм буде працювати з деякою затримкою. Тому було прийнято рішення зчитувати деяку кількість кадрів першим потоком, після чого чекати, поки другий потік обробить цю кількість кадрів.

Схему алгоритму обробки відеопотоку в реальному часі, зображено на рисунку 3.2

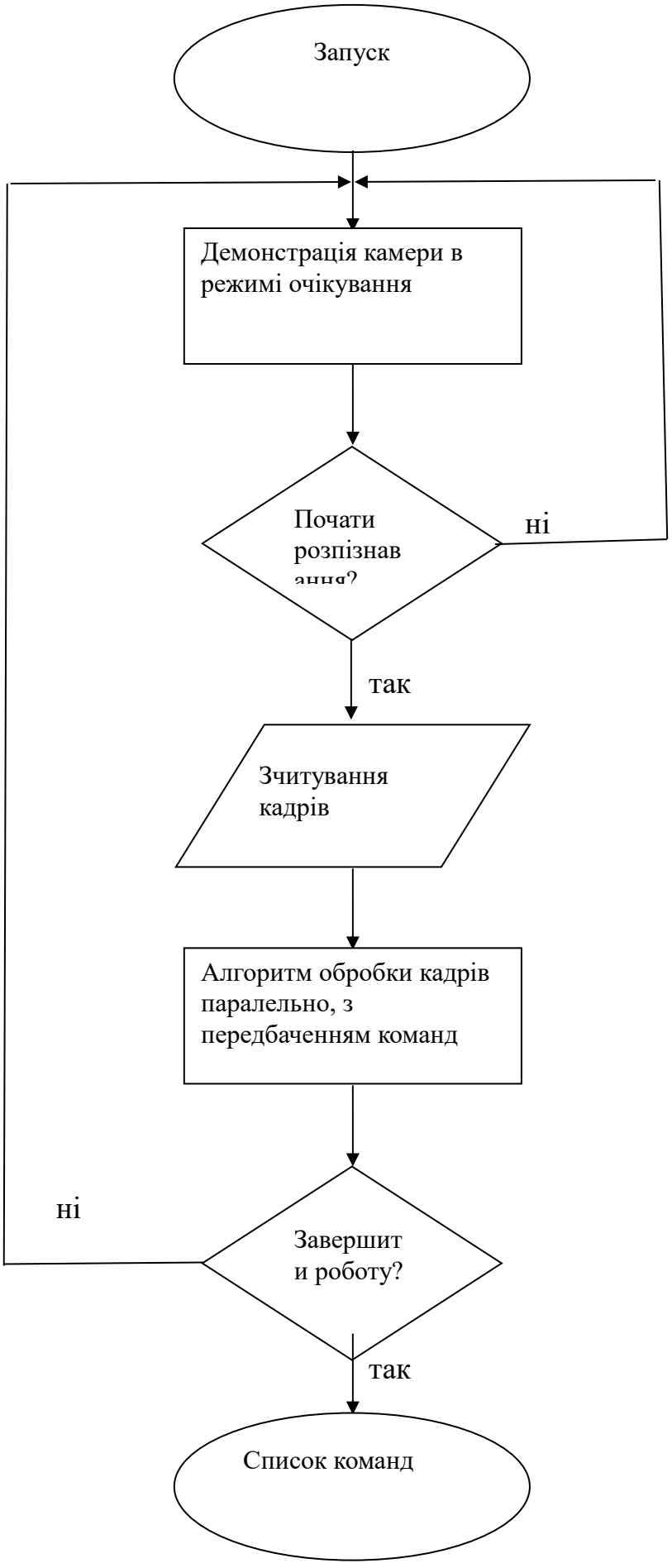

Запуск
Демонстрація камери в режимі очікування
Почати розпізнавання?
ні
так
Зчитування кадрів
Алгоритм обробки кадрів паралельно, з передбаченням команд
ні
Завершити роботу?
так
Список команд

Рисунок 3.2 –Схема обробки в реальному часі

Схема потокової обробки кадрів зображено на рисунку 3.3

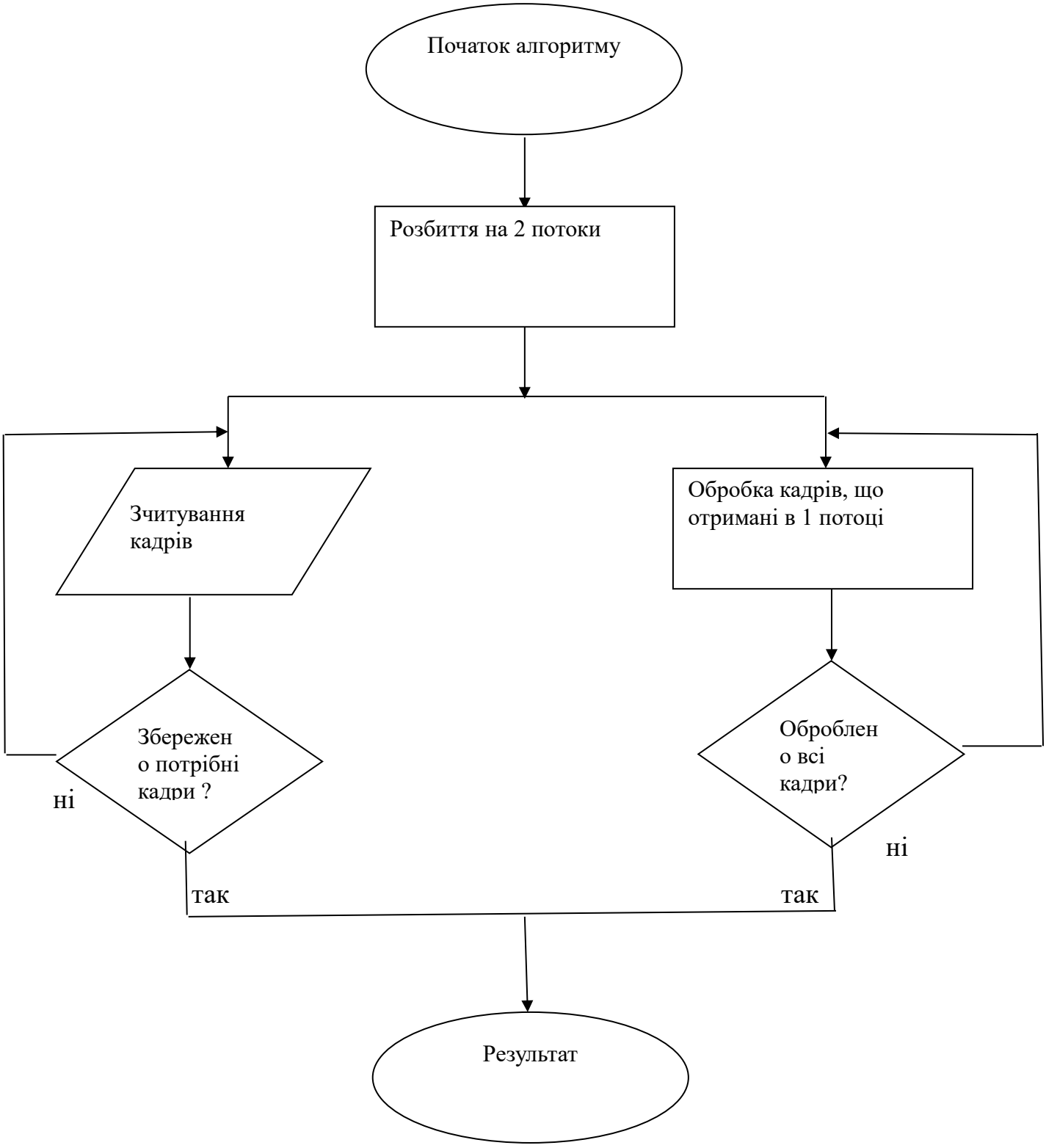


Рисунок 3.3 – Схема потокової реалізації

Режим обробки існуючого відеозапису

В даному режимі всі обчислення проходять без допомоги паралельних обчислень, після запуску програми в такому режимі, буде запромоновано ввести шлях до відеозапису у файловій системі. Якщо заданий файл існує, одразу починається алгоритм пошуку речень, тобто знаходяться паузи у вимові і відеозапис поділяється на частини, які в подальшому обробляє алгоритм розпізнавання. Схему алгоритму обробки існуючого відеозапису, зображено на рисунку 3.4

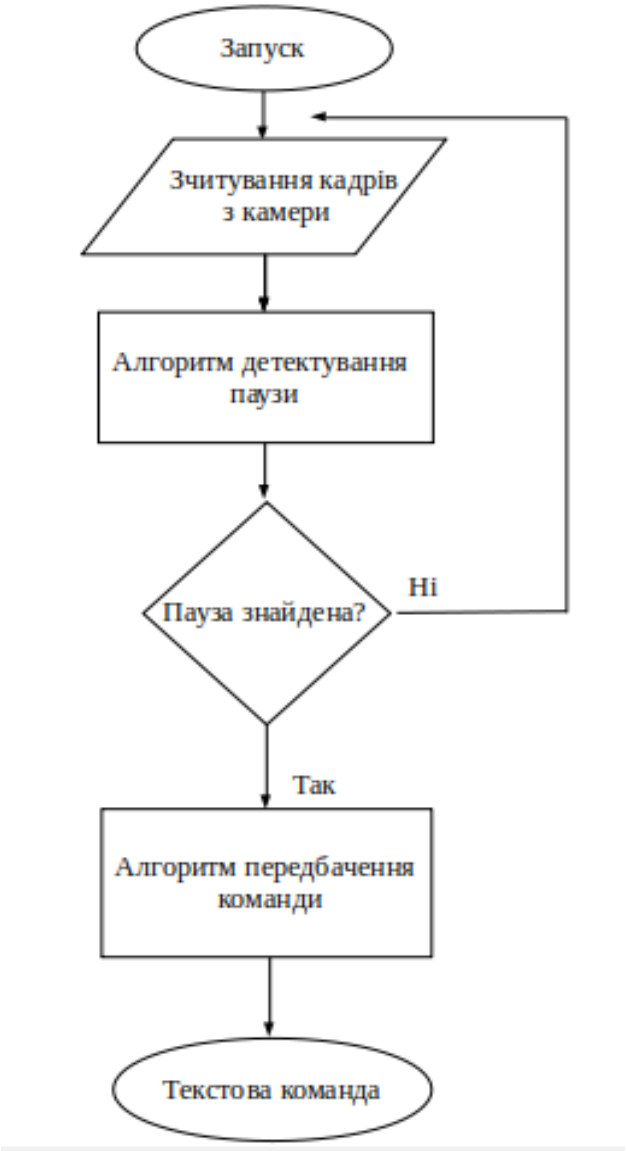


Рисунок 3.4 – Схема обробки існуючого відео

Зображення залежностей математичних величин

В режимі зображення залежностей математичних величин, демонструється графік залежності площі губ, зайнятої в кожному кадрі, та дисперсії за минулі кадри, від часу. Запропонований режим було розроблено для аналізу ознак відеозаписів.

3.3 Керівництво користувача

Після запуску програми, користувач має змогу обрати один з трьох подальших режимів роботи:

- режим реального часу
- режим обробки існуючого відеозапису
- зображення залежностей математичних величин

На рисунку 3.5 зображено консольний інтерфейс вибору режиму

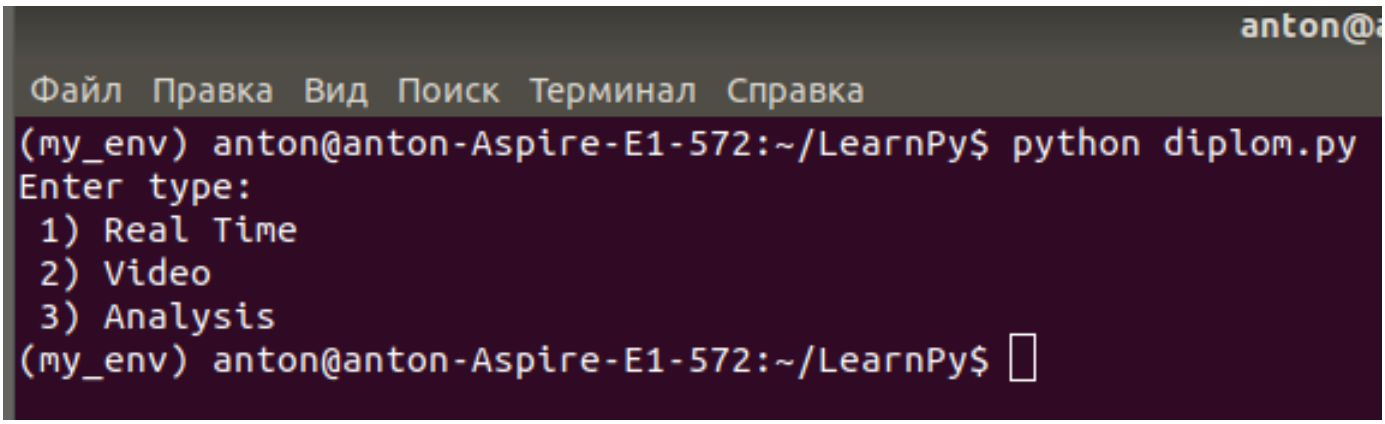


Рисунок 3.5 – Інтерфейс вибору режиму

Режим обробки існуючого відео:

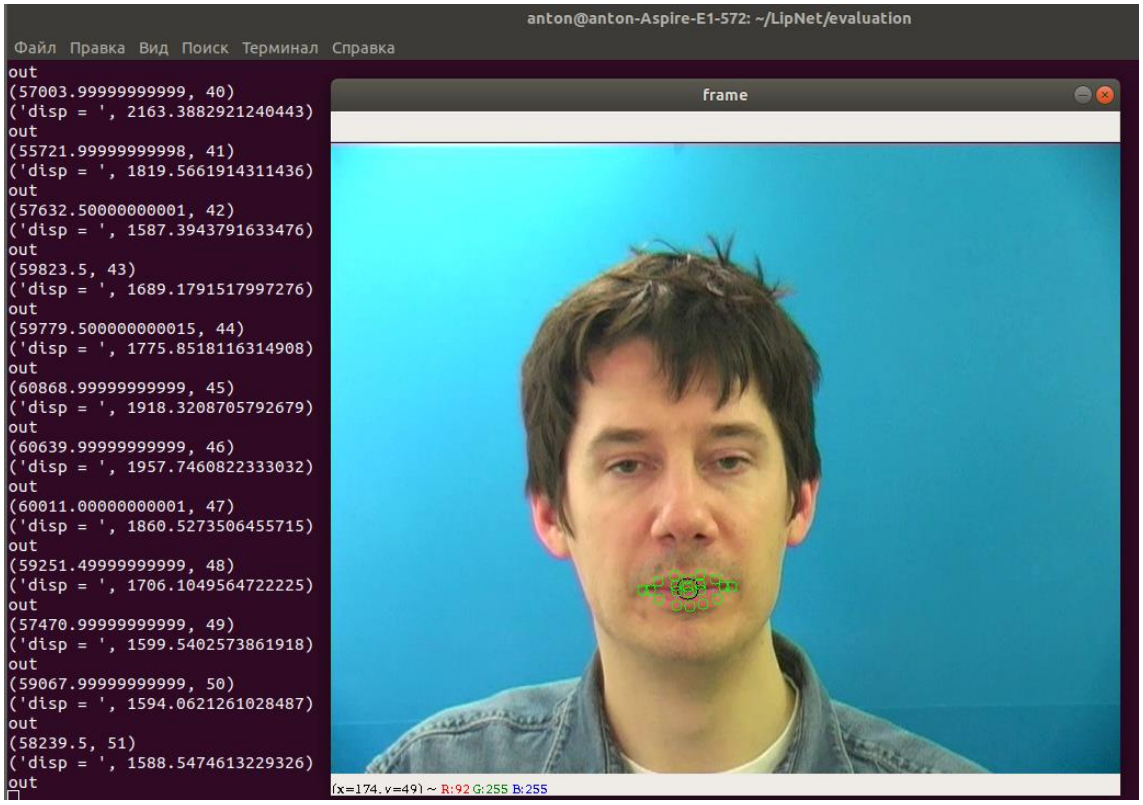


Рисунок 3.6 - Режим обробки існуючого відео

Режим залежностей характеристик:

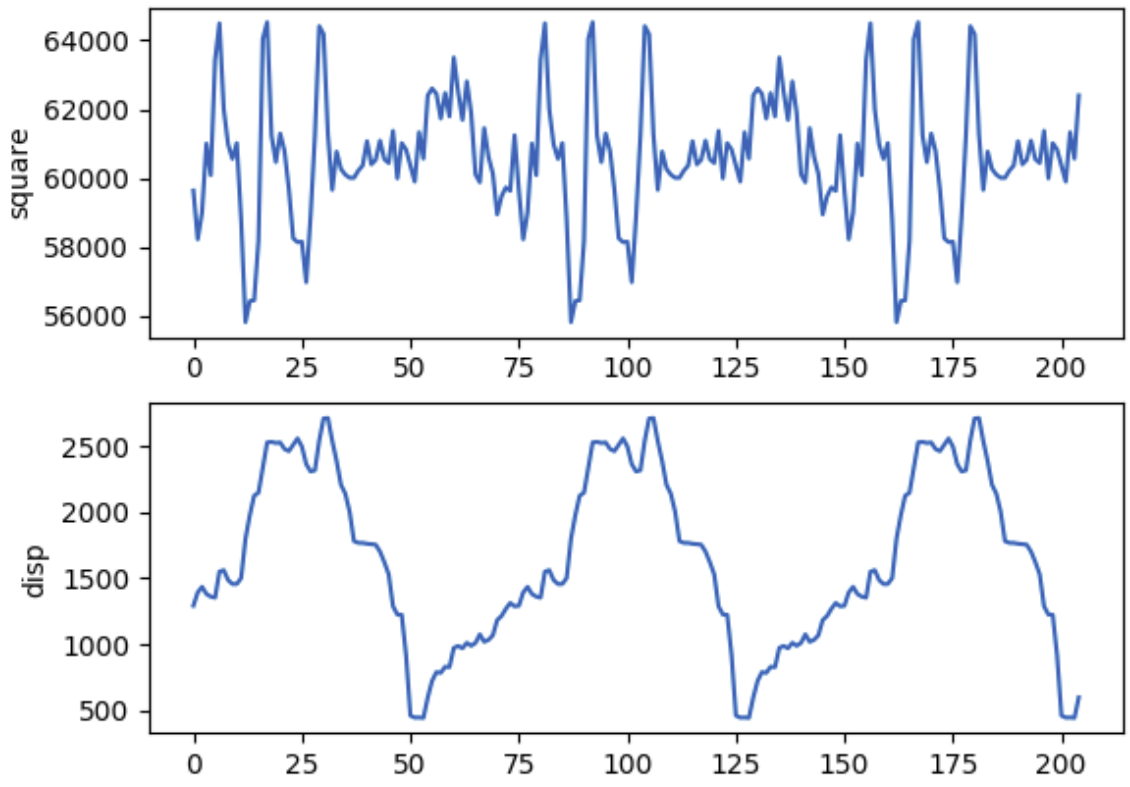


Рисунок 3.6 – Режим залежності дисперсії від часу

3.4 Аналіз результатів отриманих в роботі

Продемонструємо результати для кожного з режимів:

Режим реального часу:

Після запуску, відкривається вікно, в якому демонструється відеопотік, що транслюється з камери в даний момент часу, то пропонується натиснути на клавішу для початку розпізнавання, як показано на рисунку:

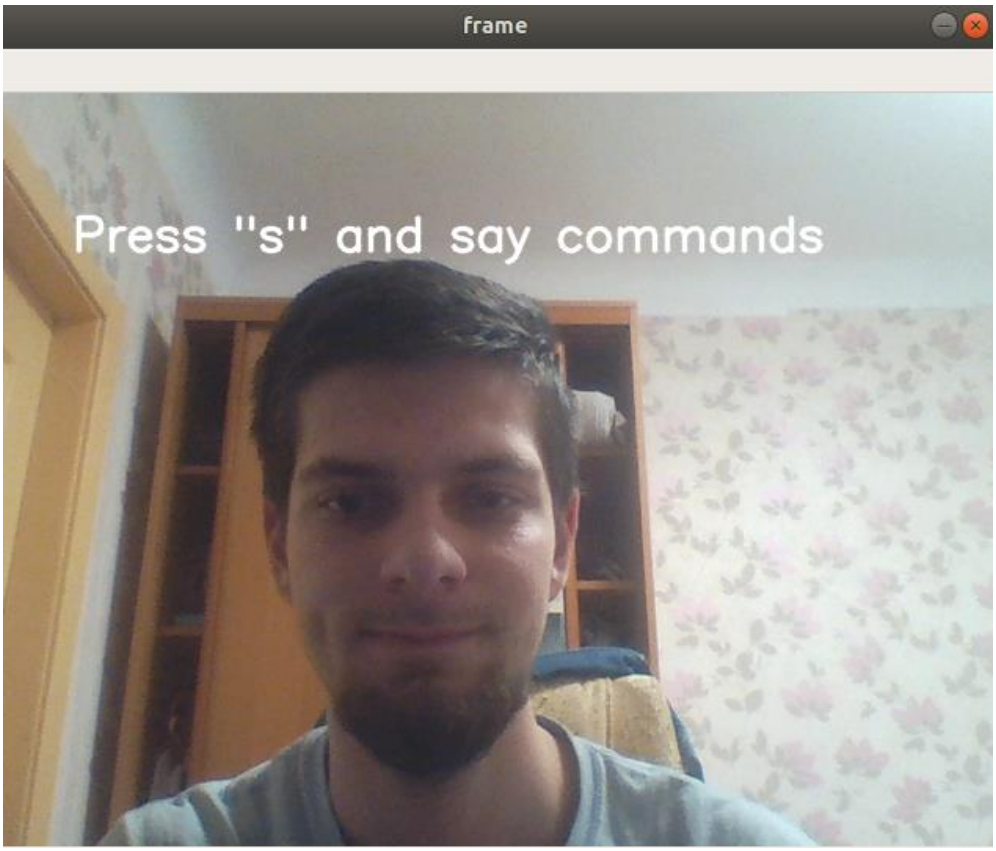


Рисунок 3.7 – Режим очікування

При натисканні на клавішу, починається алгоритм, що виконує задачі за допомогою двох потоків, як показано на рисунку 3.8

Розглянемо фразу “bin blue at a one soon”

Рисунок 3.9 – Режим запису

Оскільки потік, що зчитує кадри з відеопотоку, працює значно швидше, то після зчитування потрібної кількості кадрів даний потік завершує зберігати кадри, щоб дати можливість іншому потоку швидше обробити збережені кадри. Щоб продемонструвати це користувачу, на зображенні з’являється відповідний напис, як показано на рисунку 3.10

Рисунок 3.10 – Режим розпізнавання

Як бачимо, на рисунку 3.11, алгоритм помилився в одній букві — замість “a”, зробив передбачення в сторону букви “b” .

Рисунок 3.11 – Результат режиму реального часу

Режим аналізу:

Наведемо приклади показників для деяких відео:

а) значень площі роту

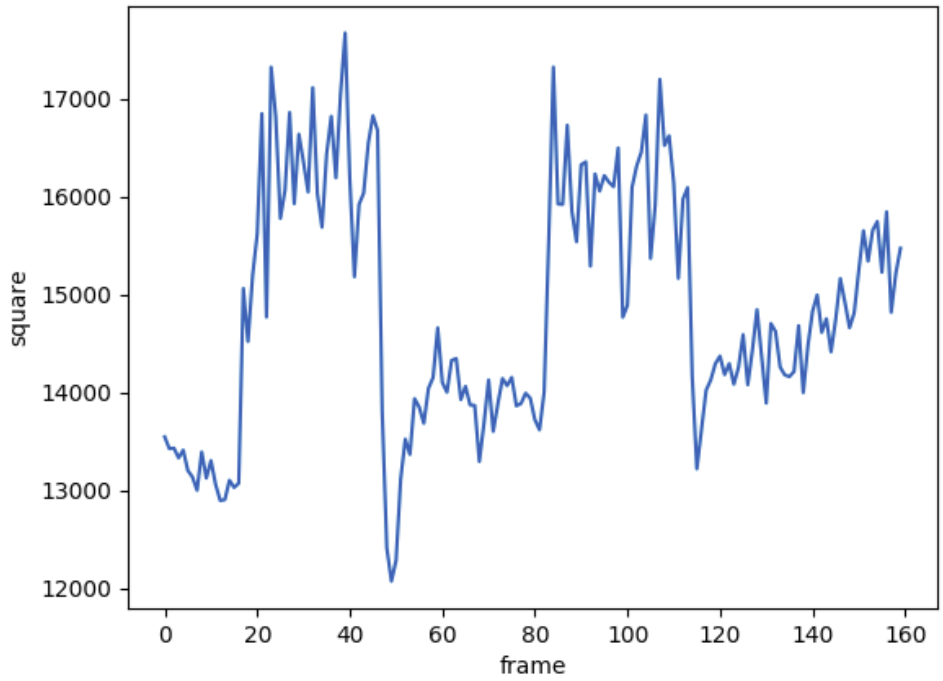


Рисунок 3.12 – Приклад залежності площі роту від часу

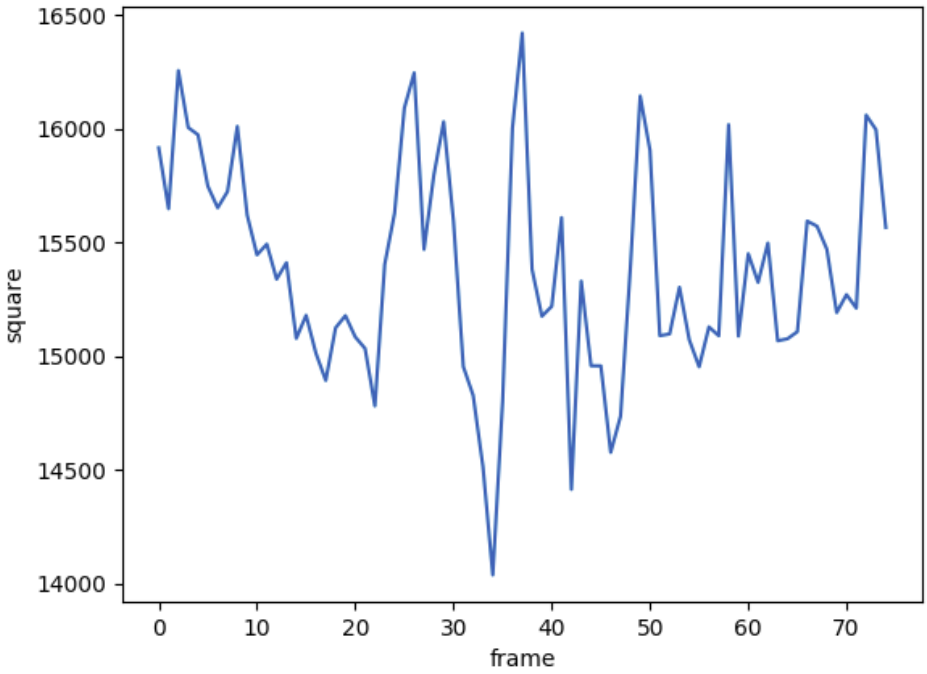


Рисунок 3.13 – Приклад залежності площі роту від часу

б) середнього значення площі роту за останні 20 кадрів

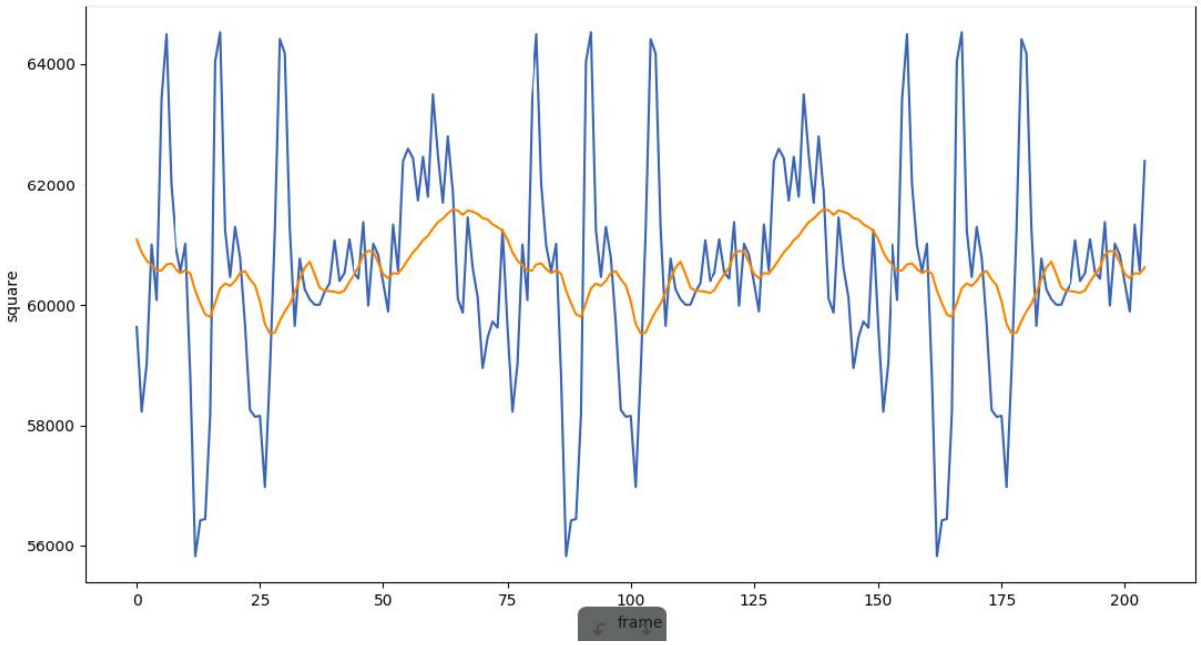


Рисунок 3.14 – Приклад залежності середнього значення площі роту від часу

в) значень дисперсії за останні 20 кадрів:

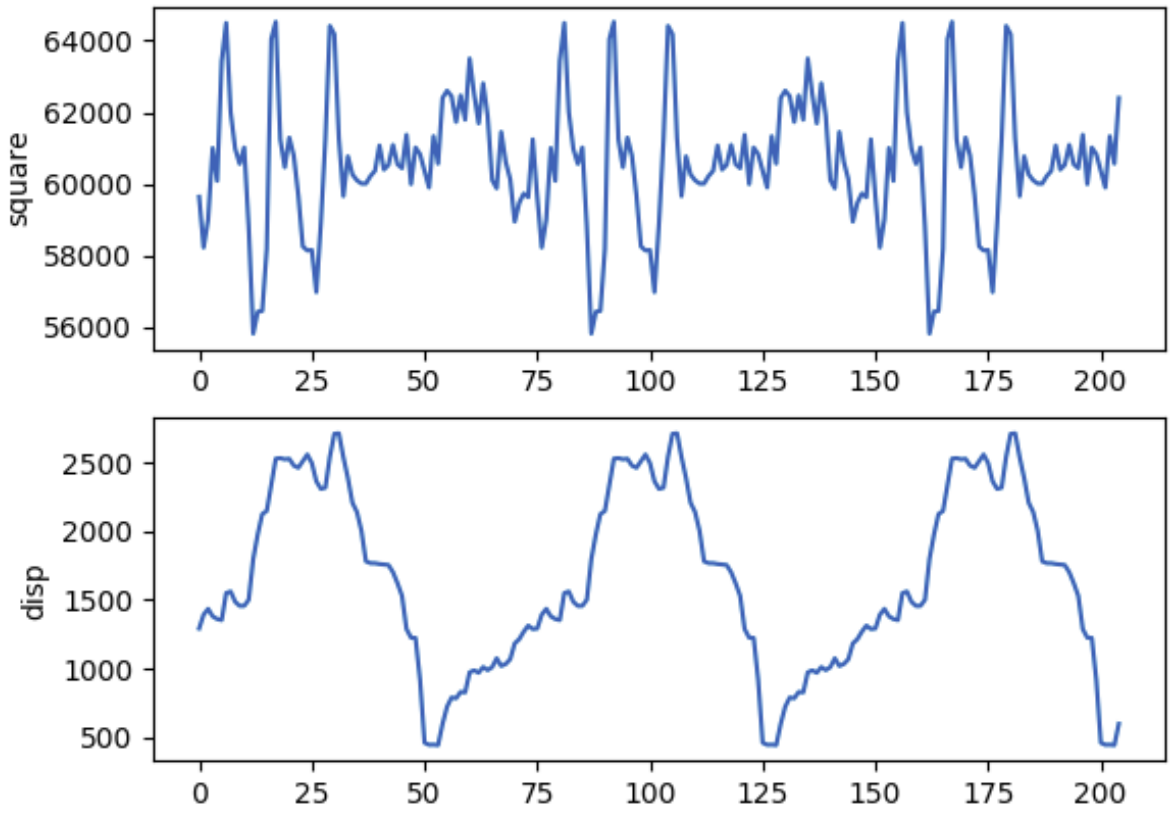


Рисунок 3.15 – Приклад залежності дисперсії площі роту від часу

По даним зображенням, можемо побачити, що дійсно найкращий спосіб для розділення мови на речення являє собою обрахунок дисперсії площі роту за деякий проміжок часу, тобто відслідковування стану спокою.

Для фрази “lay blue at q seven again”, що була сказана спікером з набору даних GRID, як показано на рисунку 3.16

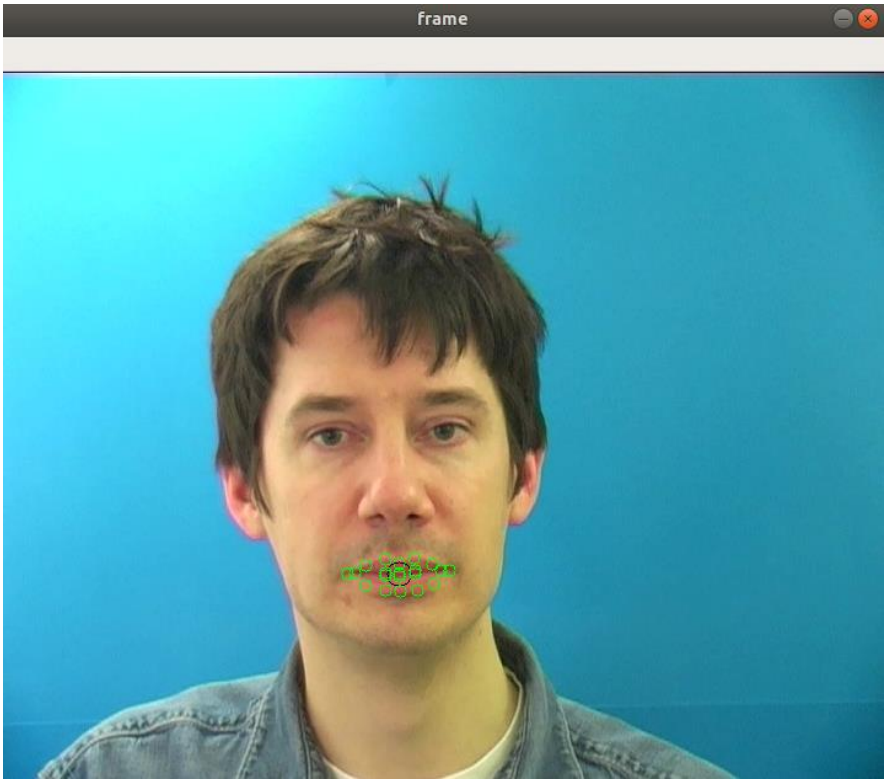


Рисунок 3.16 – Приклад запущеної програми

Програма зробила вірне припущення, як показано на рисунку 3.17

```
('RESULT: ', 'lay blue at q seven again')
```

Рисунок 3.17 – Результат для існуючого відео

3.5 Аналіз якості роботи системи

Проаналізуємо результати роботи розробленого програмного продукту. Проведемо аналіз за характеристиками:

а) Швидкодія – даний показник залежить від багатьох факторів, таких як характеристики обчислювальної машини та характеристики відеозапису. Програмний продукт протестовано на обчислювальній машині без графічного адаптера, з процесором intel i3. Такі показники в наш час не є найкращими для обробки відео, тому результати на сучасній обчислювальній машині будуть значно кращими. При розмірності зображення 600 на 800 пікселів, було досягнуто швидкодії режиму реального часу в 10-20 кадрів в секунду, що дозволяє в нормальному режимі зчитувати команди за візуальними характеристиками. Загалом оцінити швидкодію можна як достатню, з можливістю подальшого покращення.

б) Точність розпізнавання команд – для відеозаписів з набору даних, на якому тестувався програмний продукт, точність розпізнавання досягає 97 відсотків. Це є одним з найкращих показників у даній галузі. Для режиму реального часу точність є значно меншою (50-60 відсотків). Це пов’язано із швидкодією обробки кадрів, яка є меншою за 25 кадрів за секунду, та з тим що модель навчалася на людях з набору даних GRID. Тобто можна зробити висновок, що також багато залежить від вимови спікера.

Отже, проаналізувавши головні характеристики програмного продукту, головним висновком є можливість використання системи. Продемонстрований програмний продукт є унікальним, так як не існує аналогічних готових рішень, та є готовим до використання.

Недоліками є достатньо мала кількість слів для розпізнавання - 51. Такої кількості слів вистачає для роботи з командами, але при подальших покращеннях та дослідженнях цього занадто мало. Тож головним шляхом для покращення програмного продукту є збільшення словника.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Постановка задачі техніко-економічного аналізу

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для аналізу якості зображень. Інтерфейс користувача був розроблений за допомогою мови програмування Python у середовищі розробки Sublime Text 2. Інтерфейс користувача створений за допомогою технології HTML.

Програмний продукт призначено для використання на персональних комп’ютерах під управлінням операційної системи Windows, Linux, OS X.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

– визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі

оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

– для кожної функції визначаються повні річні витрати й кількість робочих часів.

– для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

– після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу системи розпізнавання команд за допомогою читання з губ. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

– програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;

– забезпечувати високу швидкість обробки великих об’ємів даних (зображень) у реальному часі;

– забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;

– передбачати мінімальні витрати на впровадження та використання програмного продукту.

Головна функція F_0– розробка програмного продукту, який аналізує зображення за допомогою нейронної мережі. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір мови програмування;

F_2 – розпізнавання якостей зображення;

F_3 – інтерфейс користувача;

F_4 – збереження класифікатора;

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) мова програмування Python;

б) мова програмування C#;

Функція F_2:

а) використання RandomForest;

б) використання нейронної мережі.

Функція F_3:

а) інтерфейс користувача, створений за технологією HTML;

б) інтерфейс користувача, створений за технологією Windows Forms.

Функція F_4:

а) збереження класифікатора в файл;

б) збереження класифікатора в оперативній пам’яті.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (табл. 4.1).

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

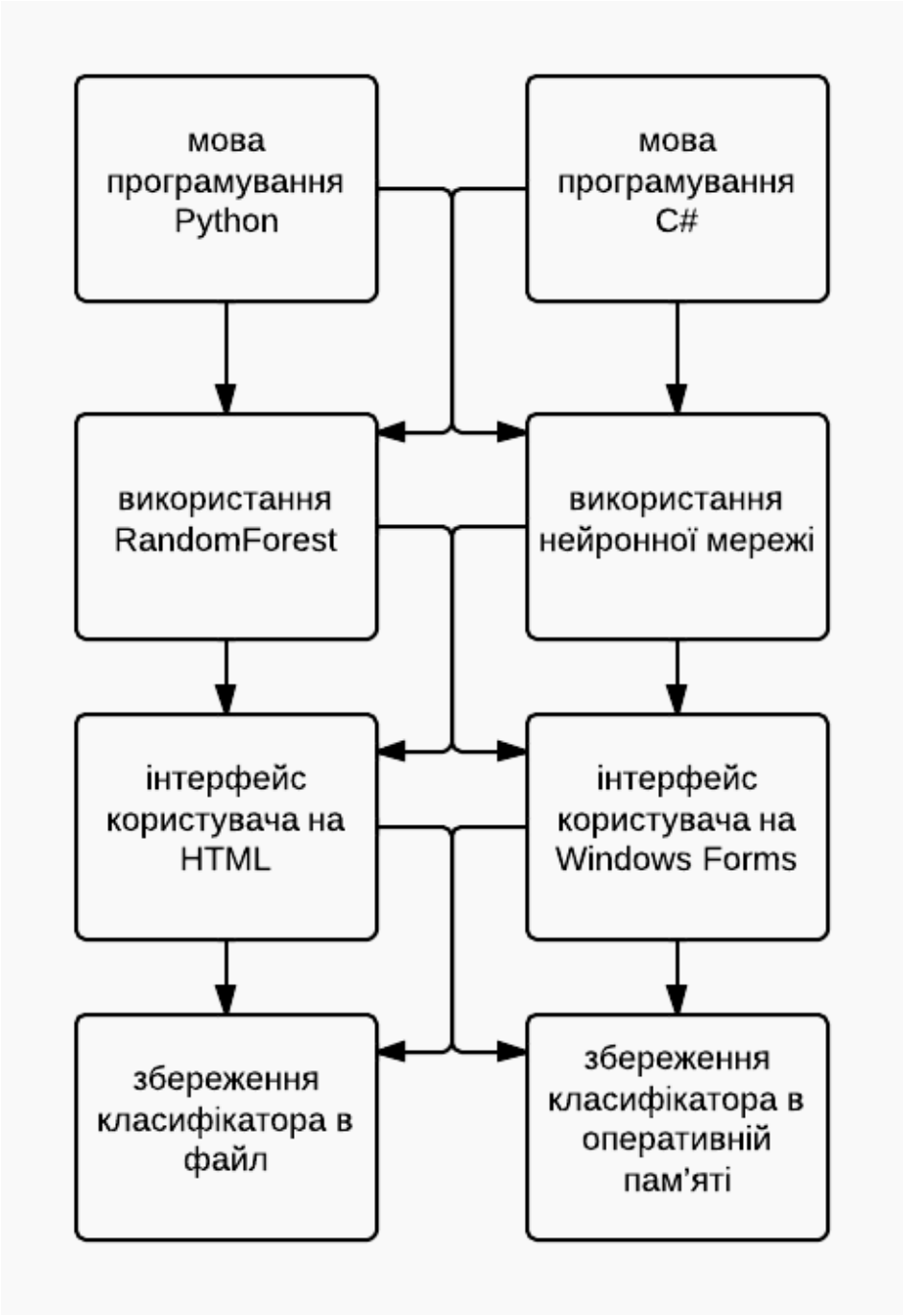


Рисунок 4.1 – Морфологічна карта

Функція *F1:*

Оскільки розрахунки можуть проводитися на сервері, час виконання не є суттєвим, а час розробки є дуже обмеженим зі складними алгоритмам, тому б) має бути відкинутий.

Функція *F2:*

Оскільки без чітко відокремлених ознак, програмний продукт не буде виконувати свою головну мету, то варіант а) відкидається.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	*А*	Займає менше часу при написанні коду, багато готових алгоритмів	Не дуже швидкий
	Б	Код швидко виконується	Займає значно більше часу при написанні коду
F2	А	Легко і швидко реалізується	Погана ступінь відокремлення ознак
	Б	Може давати дуже добрі результати	Треба багато часу на навчання мережі
F3	*А*	Можна зробити веб інтерфейс який не потребує встановлення програмного забезпечення	Розробка займає трохи більше часу
	Б	Можна використовувати як самостійну програму	Відсутність кросплатформеності, низька переносимість
F4	*А*	Можна навчати один раз і сберігати в файлі	Займає місце на диску
	Б	Можна навчати кожний раз і не витрачати місце на диску	Кожен раз витрачається час на навчання

Функція *F3:*

Інтерфейс користувача не відіграє велику роль у даному програмному продукті, тому вважаємо варіанти а) та б) гідними розгляду.

Функція *F4:*

Оскільки не можна знехтувати часом навчання, то краще зберігати класифікатор в файлі, а варіант б) відкидається.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2б – F3а – F4а
2. F1а – F2б – F3б – F4а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обґрунтування системи параметрів програмного продукту

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

– *X1* – швидкодія мови програмування;

– *X2* – об’єм пам’яті для збереження зображень;

– *X3* – час обробки зображення;

– *X4* – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій залежно від обраної мови програмування.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на обробку зображення.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.2.

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2-рис. 4.5.

Таблиця 4.2 – Основні параметри ПП

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	Оп/мс	10^6	10^7	10^8
Об’єм пам’яті для збереження даних	X2	Мб	20	15	5
Час обробки даних алгоритмом	X3	мс	6000	3000	600
Потенційний об’єм програмного коду	X4	кількість строк коду	3000	2500	1500

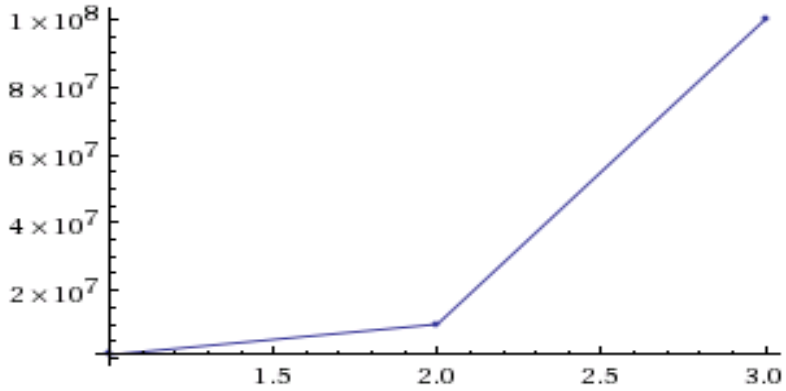


Рисунок 4.2 – X1, швидкодія мови програмування

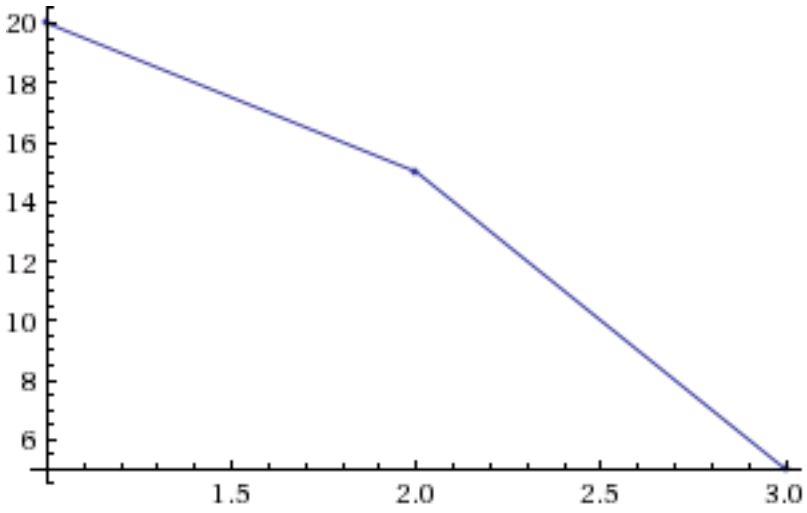


Рисунок 4.3 – X2, об’єм пам’яті для збереження даних

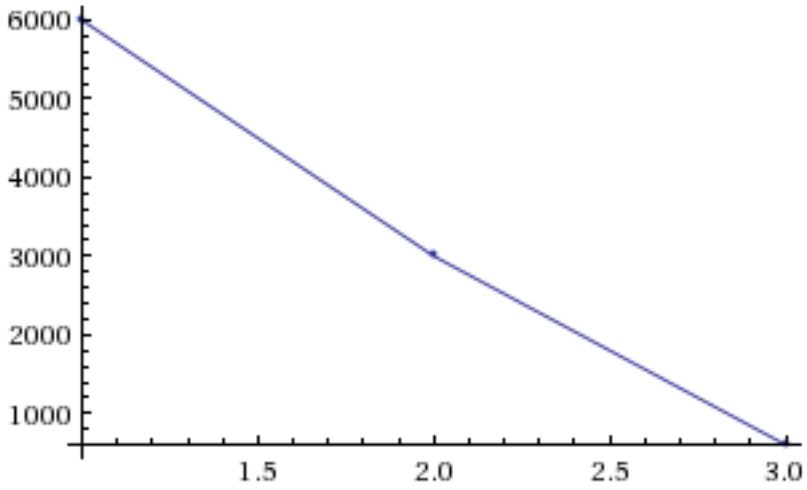


Рисунок 4.4 – X3, час обробки даних алгоритмом

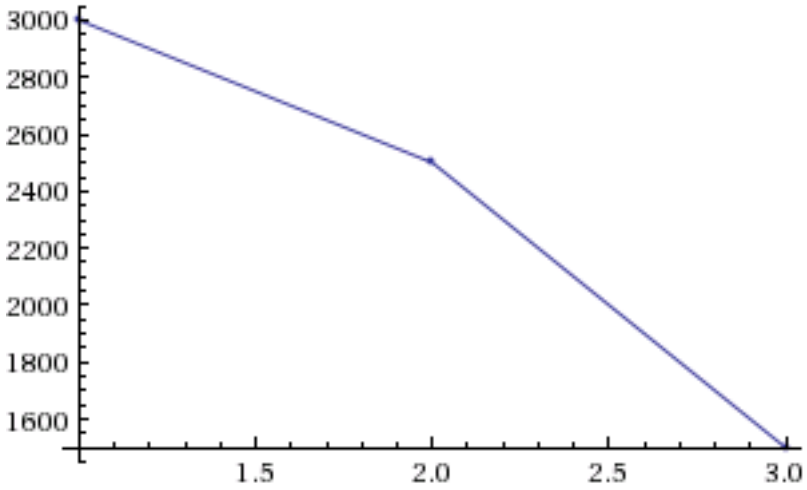


Рисунок 4.5 – X4, потенційний об’єм програмного коду

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні оцінки ознакам якості зображень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	3	3	4	3	1	2	1	17	-0.5	0.25
X2	Об’єм пам’яті для збереження даних	Мб	2	2	1	2	2	3	2	14	-3.5	12.25
X3	Час обробки даних алгоритмом	Мс	1	1	2	1	3	4	3	15	-2.5	6.25
X4	Потенційний об’єм програмного коду	кількість строк коду	4	4	3	4	4	1	4	24	6.5	42.25
	Разом		10	10	10	10	10	10	10	70	0	61

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17.5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 61.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 61}{7^2(4^3 - 4)} = 0{,}76 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю4.4.

Таблиця4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	>	>	<	<	<	>	1,5
X1 і X3	>	>	>	>	<	<	<	>	1,5
X1 і X4	<	<	>	<	<	>	<	<	0,5
X2 і X3	>	>	<	>	>	>	<	<	0,5
X2 і X4	<	<	<	<	<	>	<	<	0,5
X3 і X4	<	<	<	<	<	>	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij}визначається по формулі:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i}$,де $b_i = \sum_{i=1}^{N} a_{ij}$.

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$K_{\text{вi}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}$,де $b_i' = \sum_{i=1}^{N} a_{ij} b_j$.

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметриx_i	Параметриx_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{вi}}$	b_i^1	$K_{\text{вi}}^1$	b_i^2	$K_{\text{вi}}^2$
X1	1,0	1,5	1,5	0,5	2,5	0,129	24,25	0,215	122.1	0,316
X2	1,5	1,0	0,5	0,5	3,5	0,312	21,25	0,121	119.32	0,187
X3	1,5	1,5	1,0	0,5	4,5	0,114	29,25	0,243	145.51	0,294
X4	0,5	0,5	0,5	1,0	2,5	0,213	17,25	0,371	72.12	0,133
Всього:					13	1	81	1	412	1

4.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2*(об’єм пам’яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 6000 мс або варіанту б) 600мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів;$K_{вi}$– коефіцієнт вагомості i–го параметра;B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	10^8	2.9	0.316	0,9164
F2(X2)	А	5	3.2	0.187	0,5984
F4(X1)	А	10^8	2.8	0.316	0.8848
F3(X3,X4)	А	6000	2.1	0.294	0,6174
	Б	600	1.1	0.133	0,1463

За даними з таблиці 4.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

K_{K1} = 0,9164 + 0,5984 + 0,8848 + 0.6174= 3,017

K_{K2} = 0,9164 + 0,5984 + 0,8848 + 0.1463 = 2,5459

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М},$$

де T_P – трудомісткість розробки ПП;$K_П$ – поправочний коефіцієнт;$K_{СК}$ – коефіцієнт на складність вхідної інформації; K_M – коефіцієнт рівня мови програмування;$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це

за допомогою коефіцієнта $К_{СТ}$ = 0.7. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$Т_1$ = 90·1.7·0.7 = 107.1 людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =27 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.7:

$Т_2$ = 27 · 0.9 · 0.7 = 17.01 людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$Т_I$ = (107.1 + 17.01 + 4.87 + 17.01) · 8 = 1167.92 людино-годин;

$Т_{II}$ = (107.1 + 17.01 + 3.12 + 17.01) · 8 = 1153.92 людино-годин;

Найбільш високу трудомісткість має варіант I.

В розробці беруть участь два програмісти з окладом 12000 грн., один фінансовий аналітик з окладом 8000грн. Визначимо зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.},$$

де М – місячний оклад працівників;T_m – кількість робочих днів тиждень;t – кількість робочих годин в день.

$$С_Ч = \frac{12000 + 12000 + 8000}{3 \cdot 21 \cdot 8} = 63.5 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$С_{ЗП} = С_Ч \cdot Т_i \cdot К_Д$,

де $С_Ч$– величина погодинної оплати праці програміста;$Т_i$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП}$ = 63.5 · 1167.92 · 1.2 = 88995.5 грн.

II. $С_{ЗП}$ = 63.5 · 1153.92 · 1.2 = 87928.7 грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 88995.5 \cdot 0.22 = 19579.01$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 87928.7 \cdot 0.22 = 19344.314$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 12000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 12000 \cdot 0,2 = 28800$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 28800 \cdot (1 + 0.2) = 34560$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 34560 \cdot 0,22 = 7603.00$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 8000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 8000 = 2300$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 8000 \cdot 0.05 = 460$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706.4$ годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706,4 \cdot 0,156 \cdot 0,2436 \cdot 2,7515 = 228,74818$ грн.,

де N_C – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР}·0.67 = 8000· 0,67 =5360$ грн.

Тоді, річні експлуатаційні витрати будуть:

$С_{ЕКС} =С_{ЗП}+ С_{ВІД}+ С_А + С_Р+ С_{ЕЛ} + С_Н$

$С_{ЕКС} = 34560 + 7603 + 2300 + 460 + 228.75 + 5360 = 50511.75$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М-Г} = С_{ЕКС}/ Т_{ЕФ} = 50511.75/1706.4 = 29.6$ грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$С_М = С_{М-Г}·Т$

I. $С_М = 29.6·1167.92 = 34570.432$ грн.;

II. $С_М = 29.6·1153.92 = 34156.032$ грн.;

Накладні витрати складають 67% від заробітної плати:

$С_Н = С_{ЗП}· 0,67$

I. $С_Н = 88995.5 · 0,67 = 59626.985$ грн.;

II. $С_Н = 87928.7 · 0,67 = 54892.229$грн.;

Отже, вартість розробки ПП за варіантами становить:

$С_{ПП} = С_{ЗП}+ С_{ВІД}+ С_М +С_Н$

I. $С_{ПП} = 88995.5 + 7603 + 34570.432+ 59626.985 = 190795.917$ грн.;

II. $С_{ПП} = 87928.7 + 7603 + 34156.032 + 54892.229= 184579.961$грн.;

4.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$К_{ТЕРj} =К_{Кj}/С_{Фj}$,

$К_{ТЕР1} = 3.017 / 190795.917= 16·10^{-6}$;

$К_{ТЕР2} = 2.5459 / 184579.961= 14·10^{-6}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР2}= 16\cdot 10^{-6}$.

4.6 Висновки до розділу

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 16 \cdot 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – Python;

– використання нейронної мережі;

– інтерфейс користувача, створений за технологією HTML;

– збереження класифікатора в файлі;

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

ВИСНОВКИ

У наш час, багато програмних додатків потребують використання відео обробки для вирішення задач. Однією з таких задач є розпізнавання мови без використання аудіо-сигналу. В даній дипломній роботі запропоновано підхід для вирішення такого типу задач. В роботі було розглянуто такі питання:

а) актуальність задачі

б) вимоги до роботи, та програмного продукту

в) розроблено додаток, що працює в режимі реального часу

г) розроблено алгоритм детекції пауз між різними фразами

д) реалізовано інтерфейс користувача для даного додатку

є) проаналізовано існуючі моделі розпізнавання

ж) проведення дослідження набору даних

з) дослідження характеристик відео в режимі реального часу

Розроблено програмний продукт з такими функціями:

а) розпізнавання обличчя

б) розпізнавання губ на обличчі

в) обрахунок площі на зображенні, що займають губи

г) обчислення характеристик стану губ (в спокої, або ні)

д) алгоритм детектування пауз

є) застосування моделі для розпізнавання речень з отриманих кадрів

Проаналізувавши існуючі роботи в даному напрямку, можна зробити висновок, що в наш час, представлена робота є актуальною, так як на даний момент часу проводиться багато досліджень з питань розпізнавання мови, використовуючи лише відеопотік. Дана галузь є достатньо новою, тому в світі не існує повністю готових рішень для розпізнавання мови, використовуючи читання з губ. Отже, додаток, розроблений в даній дипломній роботі є унікальним, та є рішенням для багатьох сучасних проблем, таких як

розпізнавання мови в зашумлених місцях (заводи, фабрики), допомога людям з вадами, способи вводу та навігації.

Недоліками роботи є достатньо мала кількість слів у словнику, та невелика швидкість обробки кадрів в режимі реального часу.

Робота має багато шляхів для подальшого дослідження та розробки нових модулів. Є потреба збільшити кількість слів, доступних для розпізнавання. Одним з шляхів вдосконалення є впровадження продемонстрованого алгоритму в доповнену реальність, так як дана галузь активно розвивається в наш час.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jaimes, A.; Sebe, N. Multimodal human–computer interaction: A survey. Comput. Vis. Image Underst. 2007, 108, 116–134.

2. Loomis, J.M.; Blascovich, J.J.; Beall, A.C. Immersive virtual environment technology as a basic research tool in psychology. Behav. Res. Methods Instrum. Comput. 1999, 31, 557–564.

3. Hassanat, A.B. Visual passwords using automatic lip reading. arXiv, 2014; arXiv:1409.0924.

4. Thanda, A.; Venkatesan, S.M. Multi-task learning of deep neural networks for audio visual automatic speech recognition. arXiv, 2017; arXiv:1701.02477.

5. Biswas, A.; Sahu, P.K.; Chandra, M. Multiple cameras audio visual speech recognition using active appearance model visual features in car environment. Int. J. Speech Technol. 2016, 19, 159–171.

6. Scanlon, P.; Reilly, R. Feature analysis for automatic speechreading. In Proceedings of the 2001 IEEE Fourth Workshop on Multimedia Signal Processing (Cat. No. 01TH8564), Cannes, France, 3–5 October 2001; pp. 625–630.

7. Matthews, I.; Potamianos, G.; Neti, C.; Luettin, J. A comparison of model and transform-based visual features for audio-visual LVCSR. In Proceedings of the IEEE International Conference on Multimedia and Expo, ICME 2001, Tokyo, Japan, 22–25 August 2001; pp. 825–828.

8. Aleksic, P.S.; Katsaggelos, A.K. Comparison of low-and high-level visual features for audio-visual continuous automatic speech recognition. In Proceedings of the 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, Montreal, QC, Canada, 17–21 May 2004; p. V-917.

9. Chitu, A.G.; Driel, K.; Rothkrantz, L.J. Automatic lip reading in the Dutch language using active appearance models on high speed recordings. In Proceedings of the International Conference on Text, Speech and Dialogue, Brno, Czech Republic, 6–10 September 2010; pp. 259–266.

10. Luettin, J.; Thacker, N.A.; Beet, S.W. Visual speech recognition using active shape models and hidden Markov models. In Proceedings of the 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings, Atlanta, GA, USA, 9 May 1996; pp. 817–820.

11. Shaikh, A.A.; Kumar, D.K.; Yau, W.C.; Azemin, M.C.; Gubbi, J. Lip reading using optical flow and support vector machines. In Proceedings of the 2010 3rd International Congress on Image and Signal Processing, Yantai, China,2010; pp. 327–330.

12. Puviarasan, N.; Palanivel, S. Lip reading of hearing impaired persons using HMM. Expert Syst. Appl. 2011, 38, 4477–4481.

13. Vinyals, O.; Toshev, A.; Bengio, S.; Erhan, D. Show and tell: A neural image caption generator. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 3156–3164.

14. Gers, F.A.; Schmidhuber, J.; Cummins, F. Learning to forget: Continual prediction with LSTM. In Proceedings of the 9th International Conference on Artificial Neural Networks: ICANN'99, Edinburgh, UK, 1999;pp. 228-322..

15. Wang, Y.; Huang, M.; Zhao, L. Attention-based LSTM for aspect-level sentiment classification. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin, TX, USA, 1–5 November 2016; pp. 606–615.

16. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. arXiv, 2014; arXiv:1409.1556.

17. Matthews, I.; Cootes, T.F.; Bangham, J.A.; Cox, S.; Harvey, R. Extraction of visual features for lipreading. IEEE Trans. Pattern Anal. Mach. Intell. 2002, 24, 198–213.

18. Fan, X.; Zhang, F.; Wang, H.; Lu, X. The system of face detection based on OpenCV. In Proceedings of the 2012 24th Chinese Control and Decision Conference (CCDC), Taiyuan, China, 23–25 May 2012; pp. 648–651.

19. King, D.E. Dlib-ml: A machine learning toolkit. J. Mach. Learn. Res. 2009, 10, 1755–1758.

20. Martins, A.; Astudillo, R. From softmax to sparsemax: A sparse model of attention and multi-label classification. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1614–1623.

21. Hochreiter, S.; Schmidhuber, J. Long short-term memory. Neural Comput. 1997, 9, 1735–1780.

22. Graves, A. Long short-term memory. In Supervised Sequence Labelling with Recurrent Neural Networks; Springer: Berlin/Heidelberg, Germany, 2012; pp. 37–45.

23. Cho, K.; Courville, A.; Bengio, Y. Describing multimedia content using attention-based encoder-decoder networks. IEEE Trans. Multimed. 2015, 17, 1875–1886.

24. Zhang, Y.; Pezeshki, M.; Brakel, P.; Zhang, S.; Bengio, C.L.Y.; Courville, A. Towards end-to-end speech recognition with deep convolutional neural networks. arXiv, 2017; arXiv:1701.02720.

25. Graves, A. Generating sequences with recurrent neural networks. arXiv, 2013; arXiv:1308.0850.

ДОДАТОК А ІЛЮСТРАТИВНИЙ МАТЕРІАЛ

Система розпізнавання команд за допомогою читання з губ

Виконав:
Дровальов Антон Андрійович, ІПСА, КА-55
Науковий керівник:
к.т.н доц. Дідковська Марина Віталіївна

Актуальність

- Аутентифікація
- Інтерфейси введення інформації або управління
- Діагностика в громадських місцях
- розпізнаванням мови в шумному оточенні
- обробкою фільмів та відео

Існуючі підходи

- Марковські моделі

Однак, зазвичай моделі використовуються не для знаходження послідовності сказаних слів, а для розпізнавання фонем, тобто для отримання розподілу ймовірностей вимовлених фонем в кожен момент часу.

- Нейромережі

Потребує великого набору даних для навчання. Вимагає великої обчислювальної складності, та займає час на тренування нейронної мережі.

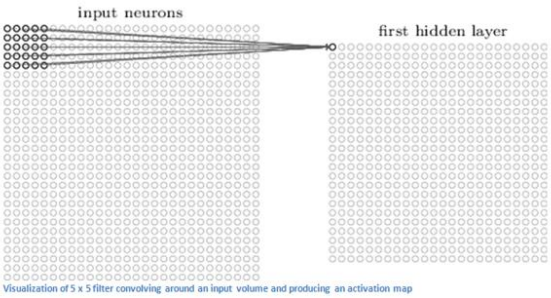
Постановка задачі

- Аналіз існуючих моделей розпізнавання
- Вибір набору даних, на якому буде проведено дослідження
- Дослідження характеристик відеозаписів даного набору даних
- Дослідження характеристик відео в режимі реального часу
- Реалізація функціоналу для програмного продукту

- Мета – розробка програмного продукту
- Предмет – методи розпізнавання команд, використовуючи лише візуальні дані
- Об’єкт – набір даних GRID містить 33 спікера, для кожного записано 1000 3-х секундних відео з вимовою 6 слів з різних класів.

command(4)+color(4)+preposition(4)+letter(25)+digit(10)+adverb(4)

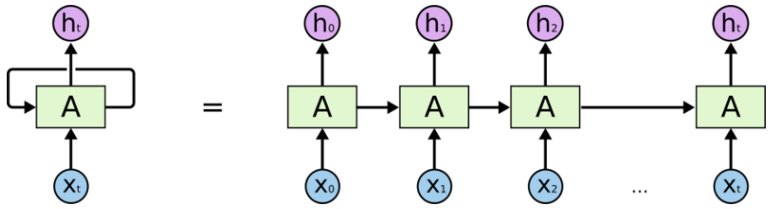
Метематичні основи (Згорткові нейромержі)



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

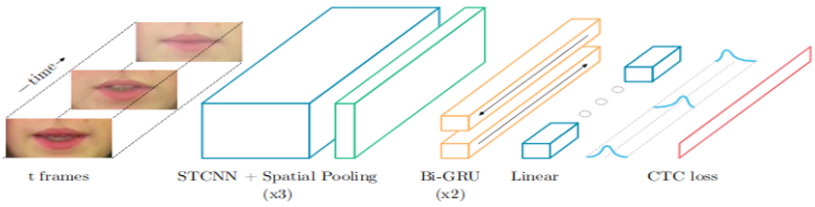
$$[\text{conv}\,(\mathrm{x},\mathrm{w})]_{c'ij} = \sum_{c=1}^{C}\sum_{i'=1}^{k_w}\sum_{j'=1}^{k_h} w_{c'ci'j'}\, x_{c,i+i',j+j'}$$

Метематичні основи (Рекурентні нейронні мережі)



Тут мережа в момент часу т приймає вхідний вектор xt, та прихований стан на попередньому кроці st-1 і обчислює вихідний вектор ht. Після цього новий стан st передається в наступну ітерацію процесу. Такі мережі дозволяють обробляти послідовності невідомої довжини, враховуючи зв'язки між теперішнім і минулим.

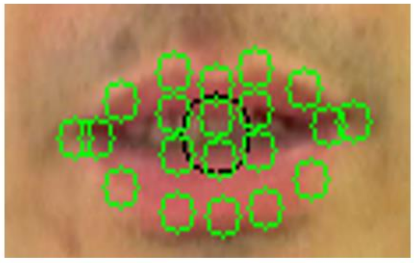
Структура застосованої нейронної мережі

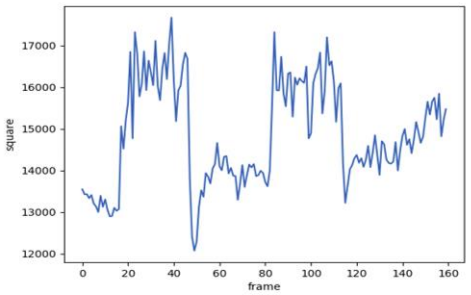


На вхід подається послідовність кадрів T, які обробляються трьома шарами згорткової нейронної мережі.Після чого витягнуті ознаки обробляються за допомогою рекурентої нейронної мережі. На виході маємо функцію активації softmax.

Алгоритм детектування пауз

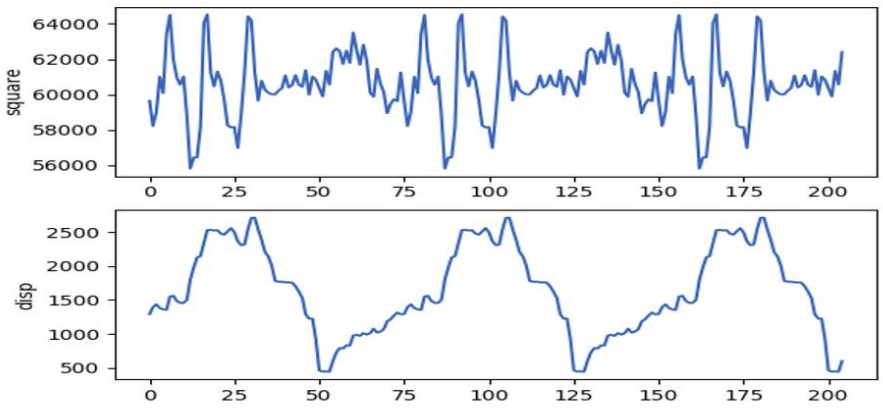
$$A = \frac{1}{2}\left|\sum_{i=1}^{n} x_i(y_{i+1} - y_{i-1})\right| = \frac{1}{2}\left|\sum_{i=1}^{n} y_i(x_{i+1} - x_{i-1})\right| = \frac{1}{2}\left|\sum_{i=1}^{n} x_i y_{i+1} - x_{i+1}y_i\right| = \frac{1}{2}\left|\sum_{i=1}^{n} det\begin{pmatrix} x_1 & y_i \\ x_{i+1} & y_{i+1} \end{pmatrix}\right|$$





Алгоритм детектування пауз

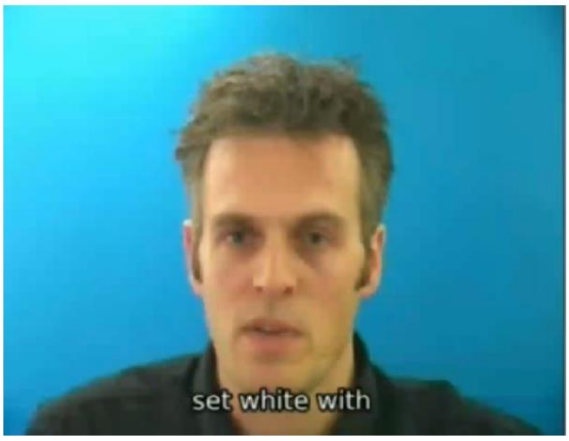
$$\sigma_y^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 = \left(\frac{1}{n}\sum_{i=1}^{n} y_i^2\right) - \bar{y}^2 = \frac{1}{n^2}\sum_{i<j}(y_i - y_j)^2$$

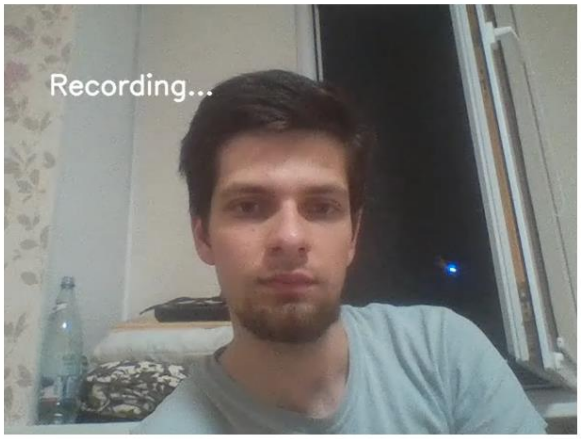


Архітектура програмного продукту

- обробка відеопотоку в реальному часі
- обробка існуючого відеозапису
- аналіз математичних показників

Приклад роботи продукту





Висновки

- Розроблено програмний продукт
- Обрана структура моделі розпізнавання
- Розроблено алгоритм поділу суцільного відеопотоку на окремі фрази

Шляхи подальшого розвитку

- Збільшення продуктивності та швидкодії
- Збільшення словнику

- Мобільні девайси
- VR/AR
- Голосовий помічник
- Зашумлені приміщення
- Охоронні системи
- Системи вводу

ДЯКУЮ ЗА УВАГУ !

ДОДАТОК Б ЛІСТИНГ ПРОГРАМИ

```
np.random.seed(55)

CURRENT_PATH = os.path.dirname(os.path.abspath(__file__))

FACE_PREDICTOR_PATH =
os.path.join(CURRENT_PATH,'..','common','predictors','shape_predictor_68_face_landmarks.dat')

PREDICT_GREEDY = False
PREDICT_BEAM_WIDTH = 200
PREDICT_DICTIONARY = os.path.join(CURRENT_PATH,'..','common','dictionaries','grid.txt')

def predict(weight_path, video_path, absolute_max_string_len=32, output_size=28):
print "\nLoading data from disk..."
video = Video(vtype='face', face_predictor_path=FACE_PREDICTOR_PATH)
if os.path.isfile(video_path):
video.from_video(video_path)
else:
video.from_frames(video_path)
print "Data loaded.\n"

#a = video.split_commands()
#show_square(video.sq)
#video.from_video_test(video_path,a)

if K.image_data_format() == 'channels_first':
img_c, frames_n, img_w, img_h = video.data.shape
else:
frames_n, img_w, img_h, img_c = video.data.shape


lipnet = LipNet(img_c=img_c, img_w=img_w, img_h=img_h, frames_n=frames_n,
absolute_max_string_len=absolute_max_string_len, output_size=output_size)

adam = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)

lipnet.model.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=adam)
lipnet.model.load_weights(weight_path)

spell = Spell(path=PREDICT_DICTIONARY)
decoder = Decoder(greedy=PREDICT_GREEDY, beam_width=PREDICT_BEAM_WIDTH,
postprocessors=[labels_to_text, spell.sentence])

X_data = np.array([video.data]).astype(np.float32) / 255
input_length = np.array([len(video.data)])

y_pred = lipnet.predict(X_data)
result = decoder.decode(y_pred, input_length)[0]

return (video, result)

if __name__ == '__main__':
if len(sys.argv) == 3:
video, result = predict(sys.argv[1], sys.argv[2])
elif len(sys.argv) == 4:
video, result = predict(sys.argv[1], sys.argv[2], sys.argv[3])
elif len(sys.argv) == 5:
video, result = predict(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4])
else:
video, result = None, ""

if video is not None:
show_video_subtitle(video.face, result)

stripe = "-" * len(result)
print " --{}- ".format(stripe)
print "[ DECODED ] |> {} |".format(result)
```

```
print " --{}- ".format(stripe)

np.random.seed(55)

CURRENT_PATH = os.path.dirname(os.path.abspath(__file__))

FACE_PREDICTOR_PATH =
os.path.join(CURRENT_PATH,'..','common','predictors','shape_predictor_68_face_landmarks.dat')



def predict(video_path):
print "\nLoading data from disk..."
video = Video(vtype='face', face_predictor_path=FACE_PREDICTOR_PATH)
if os.path.isfile(video_path):
video.from_video(video_path)
else:
video.from_frames(video_path)
print "Data loaded.\n"



a = video.split_commands()

##slide disp
d = 0
for item in video.avg_sq:
d += item
d = d/len(video.avg_sq)

print('Avarage dispertion(slide) = ', d)

#disp all
avg_sq = 0
disp = 0
for i in range(len(video.sq)):
avg_sq += video.sq[i]
disp += video.sq[i]**2

avg_sq = ((avg_sq)*(avg_sq))/len(video.sq)
disp = (disp - avg_sq)/len(video.sq)
disp = np.sqrt(disp)

print('disp = ', disp)

#avarage square
avg = 0
for item in video.sq:
avg += item
avg = avg/len(video.sq)
print('Avarage square = ', avg)




show_square(video.sq[20:],video.avg_sq)
#show_square(video.avg_sq)

#show_sq_audio(video.sq,'/home/anton/LipNet/evaluation/samples/test.wav')

#video.from_video_test(video_path,a)


if __name__ == '__main__':
if len(sys.argv) == 2:
predict(sys.argv[1])
```

```
np.random.seed(55)

CURRENT_PATH = os.path.dirname(os.path.abspath(__file__))

FACE_PREDICTOR_PATH =
os.path.join(CURRENT_PATH,'..','common','predictors','shape_predictor_68_face_landmarks.dat')
PREDICT_GREEDY = False
PREDICT_BEAM_WIDTH = 200
PREDICT_DICTIONARY = os.path.join(CURRENT_PATH,'..','common','dictionaries','grid.txt')

def process_video(weight_path,video_path):
print "\nLoading data from disk..."
video = Video(vtype='face', face_predictor_path=FACE_PREDICTOR_PATH)
if os.path.isfile(video_path):
video.from_video(video_path)
else:
video.from_frames(video_path)
print "Data loaded.\n"

a = video.split_commands()
show_square(video.sq[20:],video.avg_sq)
ans_v = []
ans_r = []

if (a!=[]):
for i in range(len(a)):
if( i == 0):
video.from_video_test(video_path,0,a[i])
v,r = predict_videos(video,weight_path)
ans_v.append(v)
ans_r.append(r)

if(i==len(a)-1):
video.from_video_test(video_path,a[i],-1,last=True)
v,r = predict_videos(video,weight_path)
ans_v.append(v)
ans_r.append(r)
break

video.from_video_test(video_path,a[i],a[i+1])
v,r = predict_videos(video,weight_path)
ans_v.append(v)
ans_r.append(r)
return ans_v,ans_r

def predict_videos(video,weight_path, absolute_max_string_len=32, output_size=28):
if K.image_data_format() == 'channels_first':
img_c, frames_n, img_w, img_h = video.data.shape
else:
frames_n, img_w, img_h, img_c = video.data.shape


lipnet = LipNet(img_c=img_c, img_w=img_w, img_h=img_h, frames_n=frames_n,
absolute_max_string_len=absolute_max_string_len, output_size=output_size)

adam = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)

lipnet.model.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=adam)
lipnet.model.load_weights(weight_path)

spell = Spell(path=PREDICT_DICTIONARY)
decoder = Decoder(greedy=PREDICT_GREEDY, beam_width=PREDICT_BEAM_WIDTH,
postprocessors=[labels_to_text, spell.sentence])

X_data = np.array([video.data]).astype(np.float32) / 255
input_length = np.array([len(video.data)])

y_pred = lipnet.predict(X_data)
result = decoder.decode(y_pred, input_length)[0]
```

```
return (video, result)


def predict(weight_path, video_path, absolute_max_string_len=32, output_size=28):
print "\nLoading data from disk..."
video = Video(vtype='face', face_predictor_path=FACE_PREDICTOR_PATH)
if os.path.isfile(video_path):
video.from_video(video_path)
else:
video.from_frames(video_path)
print "Data loaded.\n"

a = video.split_commands()
show_square(video.sq[20:],video.avg_sq)
if (a!=[]):
for i in range(len(a)):
if(i==len(a)-1):
a[i+1] = len(a)
video.from_video_test(video_path,a[i],a[i+1])


if K.image_data_format() == 'channels_first':
img_c, frames_n, img_w, img_h = video.data.shape
else:
frames_n, img_w, img_h, img_c = video.data.shape


lipnet = LipNet(img_c=img_c, img_w=img_w, img_h=img_h, frames_n=frames_n,
absolute_max_string_len=absolute_max_string_len, output_size=output_size)

adam = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)

lipnet.model.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=adam)
lipnet.model.load_weights(weight_path)

spell = Spell(path=PREDICT_DICTIONARY)
decoder = Decoder(greedy=PREDICT_GREEDY, beam_width=PREDICT_BEAM_WIDTH,
postprocessors=[labels_to_text, spell.sentence])

X_data = np.array([video.data]).astype(np.float32) / 255
input_length = np.array([len(video.data)])

y_pred = lipnet.predict(X_data)
result = decoder.decode(y_pred, input_length)[0]

return (video, result)

if __name__ == '__main__':
video = []
result = []
video,result = process_video(sys.argv[1], sys.argv[2])

list_v_r = zip(video,result)

for item in list_v_r:
if item[0] is not None:
show_video_subtitle(item[0].face, item[1])

stripe = "-" * len(item[1])
print ""
print " --{}- ".format(stripe)
print "[ DECODED ] |> {} |".format(item[1])
print " --{}- ".format(stripe)
'''
for v,r in video,result:
if v is not None:
#show_video_subtitle(v.face, r)

stripe = "-" * len(r)
```

```
print ""
print " --{}- ".format(stripe)
print "[ DECODED ] |> {} |".format(r)
print " --{}- ".format(stripe)

if len(sys.argv) == 3:
video, result = predict(sys.argv[1], sys.argv[2])
elif len(sys.argv) == 4:
video, result = predict(sys.argv[1], sys.argv[2], sys.argv[3])
elif len(sys.argv) == 5:
video, result = predict(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4])
else:
video, result = None, ""

if video is not None:
show_video_subtitle(video.face, result)

 (result)
print " --{}- ".format(stripe)
'''

class VideoCaptureAsync:
def __init__(self, src=0, width=640, height=480, nframes=120):
self.src = src
self.cap = cv2.VideoCapture(self.src)
#self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
#self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
self.grabbed, self.frame = self.cap.read()
self.nframes = nframes
self.frames = []
self.read_lock = threading.Lock()

def set(self, var1, var2):
self.capset.set(var1, var2)

def start(self):
if self.started:
print('[!] Asynchroneous video capturing has already been started.')
return None
self.started = True
self.thread = threading.Thread(target=self.update, args=())
self.thread.start()
return self

def update(self):
n = 0
while self.started:
grabbed, frame = self.cap.read()
if grabbed:
with self.read_lock:
self.frames.append(frame)
n+=1
if n>=self.nframes: break
self.started = False

def read(self,n):
frame = None
with self.read_lock:
if n<len(self.frames): frame = self.frames[n]
return frame

def stop(self):
self.started = False
self.thread.join()

def __exit__(self, exec_type, exc_value, traceback):
self.cap.release()


def test(n_frames=500, width=1280, height=720, async=False):
```

```
detector = dlib.get_frontal_face_detector()
if async:
cap = VideoCaptureAsync(0)
else:
cap = cv2.VideoCapture(0)
#cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
#cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
if async:
cap.start()
t0 = time.time()
i = 0
while i < n_frames:
_, frame = cap.read()
#dets = detector(frame, 1)
cv2.imshow('Frame', frame)
cv2.waitKey(1)
i += 1
t1 = time.time() - t0
print('[i] Frames per second: {:.2f} {}, async={}'.format(n_frames / t1, t1, async))
if async:
cap.stop()
cv2.destroyAllWindows()

def square_of_mouth(points):
det = 0
l = len(points)
for i in range(l-1):
x1 = [points[i,0],points[i,1]]
x2 = [points[i+1,0],points[i+1,1]]
a = [x1,x2]
det += abs(np.linalg.det(a))
x1 = [points[l-1,0],points[l-1,1]]
x2 = [points[0,0],points[0,1]]
det += abs(np.linalg.det(a))
return det/2

def split_dispertion(num_current,sq,window_size=20, limit=750):
disp = 0
avg_sq = 0
for i in range(num_current - window_size,num_current):
avg_sq += sq[i]
disp += sq[i]**2

#avg_sq = ((avg_sq)**2)/window_size
#disp = (disp - avg_sq)/window_size)
#disp = np.sqrt(disp)
disp = np.sqrt((disp - ((avg_sq)**2)/window_size)/window_size)

print('disp = ', disp)
#avg_sq.append(disp)
if(disp < limit):
print('added')
return num_current
else:
print('out')
return -1

def set_data(frames):
data_frames = []
for frame in frames:
frame = frame.swapaxes(0,1) # swap width and height to form format W x H x C
if len(frame.shape) < 3:
frame = np.array([frame]).swapaxes(0,2).swapaxes(0,1) # Add grayscale channel
data_frames.append(frame)
frames_n = len(data_frames)
data_frames = np.array(data_frames) # T x W x H x C
if K.image_data_format() == 'channels_first':
data_frames = np.rollaxis(data_frames, 3) # C x T x W x H
return data_frames
```

```
def predict_videos(video_data,weight_path, absolute_max_string_len=32, output_size=28):
if K.image_data_format() == 'channels_first':
img_c, frames_n, img_w, img_h = video_data.shape
else:
frames_n, img_w, img_h, img_c = video_data.shape


lipnet = LipNet(img_c=img_c, img_w=img_w, img_h=img_h, frames_n=frames_n,
absolute_max_string_len=absolute_max_string_len, output_size=output_size)

adam = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)

lipnet.model.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=adam)
lipnet.model.load_weights(weight_path)

spell = Spell(path=PREDICT_DICTIONARY)
decoder = Decoder(greedy=PREDICT_GREEDY, beam_width=PREDICT_BEAM_WIDTH,
postprocessors=[labels_to_text, spell.sentence])

X_data = np.array([video_data]).astype(np.float32) / 255
input_length = np.array([len(video_data)])

y_pred = lipnet.predict(X_data)

#print(y_pred[0,0])
#print(y_pred[0,40])
result = decoder.decode(y_pred, input_length)[0]

return result

PREDICT_GREEDY = False
PREDICT_BEAM_WIDTH = 200
CURRENT_PATH = os.path.dirname(os.path.abspath(__file__))
PREDICT_DICTIONARY = os.path.join(CURRENT_PATH,'..','common','dictionaries','grid.txt')
weight_path = 'models/overlapped-weights368.h5'
face_predictor_path = os.path.join(CURRENT_PATH,'..','common','predictors','shape_predictor_68_face_landmarks.dat')
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(face_predictor_path)

def empty_stream():
cap = VideoCaptureAsync(0)
m=0
while(True):
ret, frame = cap.read(m)
m += 1
cv2.putText(frame,'Press "s" and say commands',(50,100), cv2.FONT_HERSHEY_SIMPLEX, 1,(255,255,255),2,cv2.LINE_AA)
cv2.imshow('frame',frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
break

frames = []
lenf = 0
handling = False
mouth_frames = []
sq = []


def handle():
global handling,mouth_frames,sq
MOUTH_WIDTH = 100
MOUTH_HEIGHT = 50
HORIZONTAL_PAD = 0.19
normalize_ratio = None
min_num = 0
max_num = 0
m = 0
for i in xrange(100):
print(i)
while i>=lenf: time.sleep(0.01)
```

```
frame = frames[i]
dets = detector(frame, 1)
shape = None
for k, d in enumerate(dets):
shape = predictor(frame, d)
i = -1
if shape is None: # Detector doesn't detect face, just return as is
print('Cannot detect face')
continue

mouth_points = []
for part in shape.parts():
i += 1
if i < 48: # Only take mouth region
continue
mouth_points.append((part.x,part.y))
np_mouth_points = np.array(mouth_points)
sq.append(square_of_mouth(np_mouth_points))
print(sq[m],m)
m = m + 1

if(m>20):
max_num = split_dispertion(m,sq)
if(max_num != -1):
#split_nums.append(max_num)
print(max_num)
if(max_num - min_num > 50):
print('COMMAND = ',max_num)
video = mouth_frames[min_num:max_num]
#print(video)
video_data = set_data(video)
#print(video_data)
res = predict_videos(video_data,weight_path)
print('RESULT: ' , res)
min_num = max_num


mouth_centroid = np.mean(np_mouth_points[:, -2:], axis=0)

if normalize_ratio is None:
mouth_left = np.min(np_mouth_points[:, :-1]) * (1.0 - HORIZONTAL_PAD)
mouth_right = np.max(np_mouth_points[:, :-1]) * (1.0 + HORIZONTAL_PAD)

normalize_ratio = MOUTH_WIDTH / float(mouth_right - mouth_left)

new_img_shape = (int(frame.shape[0] * normalize_ratio ), int(frame.shape[1] * normalize_ratio))
resized_img = imresize(frame, new_img_shape)

mouth_centroid_norm = mouth_centroid * normalize_ratio

mouth_l = int(mouth_centroid_norm[0] - MOUTH_WIDTH / 2)
mouth_r = int(mouth_centroid_norm[0] + MOUTH_WIDTH / 2)
mouth_t = int(mouth_centroid_norm[1] - MOUTH_HEIGHT / 2)
mouth_b = int(mouth_centroid_norm[1] + MOUTH_HEIGHT / 2)

mouth_crop_image = resized_img[mouth_t:mouth_b, mouth_l:mouth_r]

mouth_frames.append(mouth_crop_image)
# detection end


handling = False

def stream1():
global frames, lenf, handling
cap = cv2.VideoCapture(0)
handling = False
while(True):
ret, frame = cap.read()
```

```
if ret:
if handling:
if lenf<100:
frames.append(frame)
lenf+=1
cv2.putText(frame,'Recording...',(50,100), cv2.FONT_HERSHEY_SIMPLEX, 1,(255,255,255),2,cv2.LINE_AA)
else:
cv2.putText(frame,'Recognizing...',(50,100), cv2.FONT_HERSHEY_SIMPLEX, 1,(255,255,255),2,cv2.LINE_AA)
else:
cv2.putText(frame,'Press "s" and say commands',(50,100), cv2.FONT_HERSHEY_SIMPLEX, 1,(255,255,255),2,cv2.LINE_AA)
cv2.imshow('frame',frame)
key = cv2.waitKey(1) & 0xFF
if key == ord('q'):
break
elif key == ord('s'):
handling = True
frames = []
lenf = 0
threading.Thread(target=handle).start()



def stream(detector,predictor):

cap = cv2.VideoCapture('1.mpg')
#cap = cv2.VideoCapture(0)
mouth_frames = []
sq = []
#split_nums = []
MOUTH_WIDTH = 100
MOUTH_HEIGHT = 50
HORIZONTAL_PAD = 0.19
normalize_ratio = None
min_num = 0
max_num = 0

m = 0

while(cap.isOpened):
ret, frame = cap.read()
if(m==224):
break
# detection start
dets = detector(frame, 1)
shape = None
for k, d in enumerate(dets):
shape = predictor(frame, d)
i = -1
if shape is None: # Detector doesn't detect face, just return as is
print('Cannot detect face')
continue

mouth_points = []
for part in shape.parts():
i += 1
if i < 48: # Only take mouth region
continue
mouth_points.append((part.x,part.y))
np_mouth_points = np.array(mouth_points)
'''
'''
sq.append(square_of_mouth(np_mouth_points))
print(sq[m],m)
m = m + 1

if(m>20):
max_num = split_dispertion(m,sq)
if(max_num != -1):
#split_nums.append(max_num)
print(max_num)
```

```
if(max_num - min_num > 50):
print('COMMAND = ',max_num)
video = mouth_frames[min_num:max_num]
#print(video)
video_data = set_data(video)
#print(video_data)
res = predict_videos(video_data,weight_path)
print('RESULT: ' , res)
min_num = max_num


mouth_centroid = np.mean(np_mouth_points[:, -2:], axis=0)

if normalize_ratio is None:
mouth_left = np.min(np_mouth_points[:, :-1]) * (1.0 - HORIZONTAL_PAD)
mouth_right = np.max(np_mouth_points[:, :-1]) * (1.0 + HORIZONTAL_PAD)

normalize_ratio = MOUTH_WIDTH / float(mouth_right - mouth_left)

new_img_shape = (int(frame.shape[0] * normalize_ratio ), int(frame.shape[1] * normalize_ratio))
resized_img = imresize(frame, new_img_shape)

mouth_centroid_norm = mouth_centroid * normalize_ratio

mouth_l = int(mouth_centroid_norm[0] - MOUTH_WIDTH / 2)
mouth_r = int(mouth_centroid_norm[0] + MOUTH_WIDTH / 2)
mouth_t = int(mouth_centroid_norm[1] - MOUTH_HEIGHT / 2)
mouth_b = int(mouth_centroid_norm[1] + MOUTH_HEIGHT / 2)

mouth_crop_image = resized_img[mouth_t:mouth_b, mouth_l:mouth_r]

mouth_frames.append(mouth_crop_image)
# detection end

mouth_drawed_image = frame
cv2.circle(mouth_drawed_image,(int(mouth_centroid[0]),int(mouth_centroid[1])),10,(0,0,0))
for item in np_mouth_points:
cv2.circle(mouth_drawed_image,(int(item[0]),int(item[1])),5,(0,255,0))
cv2.imshow('frame',mouth_drawed_image)
cv2.imshow('frame',frame)

if cv2.waitKey(1) & 0xFF == ord('q'):
return 1
cap.release()
cv2.destroyAllWindows()

if __name__ == '__main__':
stream(detector,predictor)
#stream1()
if __name__ == '__main__' :
detector = dlib.get_frontal_face_detector()
# Start default camera
video = cv2.VideoCapture(0)
print(video.get(cv2.CAP_PROP_FRAME_WIDTH))
print(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
# Find OpenCV version
(major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')
# With webcam get(CV_CAP_PROP_FPS) does not work.
# Let's see for ourselves.
#video.set(cv2.CAP_PROP_FPS,30)
#video.set(cv2.CAP_PROP_EXPOSURE,500)
if int(major_ver) < 3 :
fps = video.get(cv2.cv.CV_CAP_PROP_FPS)
print "Frames per second using video.get(cv2.cv.CV_CAP_PROP_FPS): {0}".format(fps)
else :
fps = video.get(cv2.CAP_PROP_FPS)
print "Frames per second using video.get(cv2.CAP_PROP_FPS) : {0}".format(fps)
exp = video.get(cv2.CAP_PROP_EXPOSURE)
print "Exposur: {0}".format(exp)
# Number of frames to capture
```

```
num_frames = 500
print "Capturing {0} frames".format(num_frames)
# Start time
start = time.time()
# Grab a few frames

#for i in xrange(0, num_frames) :
# ret, frame = video.read()
for i in xrange(0, num_frames) :
ret, frame = video.read()
cv2.imshow('frame',frame)

#dets = detector(frame, 1)

if cv2.waitKey(1) & 0xFF == ord('q'):
break
# End time
end = time.time()
# Time elapsed
seconds = end - start
print "Time taken : {0} seconds".format(seconds)
# Calculate frames per second
fps = num_frames / seconds
print "Estimated frames per second : {0}".format(fps)
# Release video
video.release()

class VideoCaptureAsync:
def __init__(self, src=0, width=640, height=480, nframes=120):
self.src = src
self.cap = cv2.VideoCapture(self.src)
#self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
#self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
self.grabbed, self.frame = self.cap.read()
self.nframes = nframes
self.frames = []
self.read_lock = threading.Lock()

def set(self, var1, var2):
self.capset.set(var1, var2)

def start(self):
if self.started:
print('[!] Asynchroneous video capturing has already been started.')
return None
self.started = True
self.thread = threading.Thread(target=self.update, args=())
self.thread.start()
return self

def update(self):
n = 0
while self.started:
grabbed, frame = self.cap.read()
if grabbed:
with self.read_lock:
self.frames.append(frame)
n+=1
if n>=self.nframes: break
self.started = False

def read(self,n):
frame = None
with self.read_lock:
if n<len(self.frames): frame = self.frames[n]
return frame

def stop(self):
self.started = False
self.thread.join()
```

```
def __exit__(self, exec_type, exc_value, traceback):
self.cap.release()


def test(n_frames=500, width=1280, height=720, async=False):
detector = dlib.get_frontal_face_detector()
if async:
cap = VideoCaptureAsync(0)
else:
cap = cv2.VideoCapture(0)
#cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
#cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
if async:
cap.start()
t0 = time.time()
i = 0
while i < n_frames:
_, frame = cap.read()
#dets = detector(frame, 1)
cv2.imshow('Frame', frame)
cv2.waitKey(1)
i += 1
t1 = time.time() - t0
print('[i] Frames per second: {:.2f} {}, async={}'.format(n_frames / t1, t1, async))
if async:
cap.stop()
cv2.destroyAllWindows()


if __name__ == '__main__':
test(n_frames=500, width=640, height=480, async=False)
test(n_frames=500, width=640, height=480, async=True)
class LipNet(object):
def __init__(self, img_c=3, img_w=100, img_h=50, frames_n=75, absolute_max_string_len=32, output_size=28):
self.img_c = img_c
self.img_w = img_w
self.img_h = img_h
self.frames_n = frames_n
self.absolute_max_string_len = absolute_max_string_len
self.output_size = output_size
self.build()

def build(self):
if K.image_data_format() == 'channels_first':
input_shape = (self.img_c, self.frames_n, self.img_w, self.img_h)
else:
input_shape = (self.frames_n, self.img_w, self.img_h, self.img_c)

self.input_data = Input(name='the_input', shape=input_shape, dtype='float32')

self.zero1 = ZeroPadding3D(padding=(1, 2, 2), name='zero1')(self.input_data)
self.conv1 = Conv3D(32, (3, 5, 5), strides=(1, 2, 2), kernel_initializer='he_normal', name='conv1')(self.zero1)
self.batc1 = BatchNormalization(name='batc1')(self.conv1)
self.actv1 = Activation('relu', name='actv1')(self.batc1)
self.drop1 = SpatialDropout3D(0.5)(self.actv1)
self.maxp1 = MaxPooling3D(pool_size=(1, 2, 2), strides=(1, 2, 2), name='max1')(self.drop1)

self.zero2 = ZeroPadding3D(padding=(1, 2, 2), name='zero2')(self.maxp1)
self.conv2 = Conv3D(64, (3, 5, 5), strides=(1, 1, 1), kernel_initializer='he_normal', name='conv2')(self.zero2)
self.batc2 = BatchNormalization(name='batc2')(self.conv2)
self.actv2 = Activation('relu', name='actv2')(self.batc2)
self.drop2 = SpatialDropout3D(0.5)(self.actv2)
self.maxp2 = MaxPooling3D(pool_size=(1, 2, 2), strides=(1, 2, 2), name='max2')(self.drop2)

self.zero3 = ZeroPadding3D(padding=(1, 1, 1), name='zero3')(self.maxp2)
self.conv3 = Conv3D(96, (3, 3, 3), strides=(1, 1, 1), kernel_initializer='he_normal', name='conv3')(self.zero3)
self.batc3 = BatchNormalization(name='batc3')(self.conv3)
self.actv3 = Activation('relu', name='actv3')(self.batc3)
```

```
self.drop3 = SpatialDropout3D(0.5)(self.actv3)
self.maxp3 = MaxPooling3D(pool_size=(1, 2, 2), strides=(1, 2, 2), name='max3')(self.drop3)

self.resh1 = TimeDistributed(Flatten())(self.maxp3)

self.gru_1 = Bidirectional(GRU(256, return_sequences=True, kernel_initializer='Orthogonal', name='gru1'),
merge_mode='concat')(self.resh1)
self.gru_2 = Bidirectional(GRU(256, return_sequences=True, kernel_initializer='Orthogonal', name='gru2'),
merge_mode='concat')(self.gru_1)

# transforms RNN output to character activations:
self.dense1 = Dense(self.output_size, kernel_initializer='he_normal', name='dense1')(self.gru_2)

self.y_pred = Activation('softmax', name='softmax')(self.dense1)

self.labels = Input(name='the_labels', shape=[self.absolute_max_string_len], dtype='float32')
self.input_length = Input(name='input_length', shape=[1], dtype='int64')
self.label_length = Input(name='label_length', shape=[1], dtype='int64')

self.loss_out = CTC('ctc', [self.y_pred, self.labels, self.input_length, self.label_length])

self.model = Model(inputs=[self.input_data, self.labels, self.input_length, self.label_length], outputs=self.loss_out)

def summary(self):
Model(inputs=self.input_data, outputs=self.y_pred).summary()

def predict(self, input_batch):
return self.test_function([input_batch, 0])[0] # the first 0 indicates test

@property
def test_function(self):
# captures output of softmax so we can decode the output during visualization
return K.function([self.input_data, K.learning_phase()], [self.y_pred, K.learning_phase()])

def show_video_subtitle(frames, subtitle):
fig, ax = plt.subplots()
fig.show()

text = plt.text(0.5, 0.1, "",
ha='center', va='center', transform=ax.transAxes,
fontdict={'fontsize': 15, 'color':'white', 'fontweight': 500})
text.set_path_effects([path_effects.Stroke(linewidth=3, foreground='black'),
path_effects.Normal()])

subs = subtitle.split()
inc = max(len(frames)/(len(subs)+1), 0.01)
i = 0
img = None
for frame in frames:
sub = " ".join(subs[:int(i/inc)])

text.set_text(sub)

if img is None:
img = plt.imshow(frame)
else:
img.set_data(frame)
fig.canvas.draw()
i += 1

def show_square(sq,avg_sq):
plt.figure()
f, axes = plt.subplots(2, 1)
axes[0].plot(sq)
axes[0].set_ylabel('square')
axes[1].plot(avg_sq)
axes[1].set_ylabel('disp')
plt.show()
```